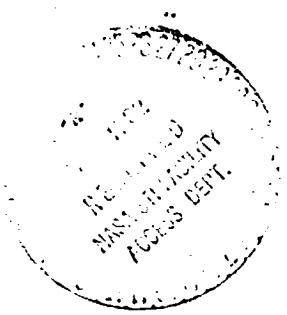
NASA

Technical Memorandum 83981

# A Fast Hidden Line Algorithm For Plotting Finite Element Models

Gary Jones



AUGUST 1982

National Aeronautics and
Space Administration

**Goddard Space Flight Center**
Greenbelt, Maryland 20771

# A FAST HIDDEN LINE ALGORITHM FOR PLOTTING FINITE ELEMENT MODELS

GARY K. JONES

NASA/GODDARD SPACE FLIGHT CENTER

AUGUST 1982

## INTRODUCTION

This report presents a very fast hidden line technique (JONES-D) developed by the author, for the interactive graphical display of NASTRAN finite element models. It is the author's opinion that hidden line plotting together with haloed line and normal (all lines visible) plotting is required for an effective finite element plot package. Illustrated in figure 1 are hidden line, haloed line, and normal plots of the same finite element model. In this example, the hidden line plot is clearly the most effective form for human understanding. Haloed plotting is the most effective with models containing no, or very few, surfaces; thereby rendering hidden line plotting inappropriate. Techniques to perform hidden line plotting have been much discussed beginning with the advent of computer graphics in the early 1960's and continuing into the present era. Given the bulk of this prior work (for example references 1 through 9), why develop a new method? The answer is that these prior methods appear to lack the speed for effective interactive use, or lack the features required to effectively plot NASTRAN models. Except for the Watkins technique (reference 9), code to implement these techniques were not published or generally available. Experience in using the Watkins technique had shown it not to be acceptable for the proposed use. References 10, 12, and 13 were published after the JONES-D method had been completed or substantially so, these new methods together with the Watkins method are compared with the technique developed herein in a subsequent section of this report.

## DESIGN GOALS

In order to understand the design goals, the computing environment must be defined. The Applied Engineering Division's (NASA Goddard Space Flight Center) computer system consists of a Digital Equipment Corporation VAX 11/780 computer and support devices. The host operating system used for developing and testing the hidden line routine was VAX VMS version 2.5. The maximum interactive user working set size was 256 kbytes. The target graphic display devices were the four Tektronix 4014 (or equivalent) terminals used by the structural analysts within the Applied Engineering Division. These terminals were connected to the host VAX via 9600 baud RS-232 data links, and the base plot software was Tektronix PLOT 10 running on the host. The specific design goals for the hidden line routine were:

a. The hidden line routine must function within the NPLOT NASTRAN plot package being developed by the author. The NPLOT program supports plotting of normally used NASTRAN elements: line elements (CBAR, CROD, etc.), surface elements (CQUAD, CTRIA, etc.) and solid elements (CHEXA and CPENTA). The hidden line technique must support these elements. The hidden line routine must also be supportive of labeling of visible grid points and elements.

b. Hidden line plotting must be as responsive to the interactive user as normal, all lines visible, plotting. This requires an elapsed time per hidden line plot of about the same as, or less than, that for a normal plot. It was recognized that this was a very ambitious goal. If we examine the data flows

associated with the normal, halo, and hidden line plotting, figure 2, we can see why this goal is at least conceptually possible and why halo line plotting is usually less responsive than normal plotting. The data flow for normal plotting consists of processing N vectors through PLOT 10 software and transmitting the appropriate commands to the graphic terminal. The halo plot module, figure 2, normally expands the N vector list into a vector list of length $\sim$ 1.5 N. Therefore, we would expect halo plotting to be more than 1.5 times slower than normal plotting. Conversely, the hidden line module, figure 2, truncates the input vector list of length N to about 0.5N. Thus, hidden line plotting would equal the responsiveness of normal plotting, provided that the hidden line module can execute in about the same time as that for processing 0.5 N vectors through the normal plot flow path.

c. A high degree of reliability is required. A plot routine that typically generates plot errors is worse than useless for the debugging of finite element models. The required error rate from the hidden line routine must be very low.

## IMPLEMENTATION

Several different variations of the same basic hidden line method were sequentially developed in the course of this effort. To keep track of the different versions, they were assigned the names JONES-A through JONES-D. The fastest and most recent version, JONES-D, is the subject of this report. The algorithm (Appendix B) was written in DEC VAX FORTRAN 77 and embedded in the

NPLOT NASTRAN plot program. This implementation made considerable use of the VAX virtual memory feature to maximize performance. The basic flow of the technique is as follows:

o INPUT: The chief inputs to JONES-D from NPLOT are the global edge vector list, global surface list and grid point table. It should be noted that NPLOT operates to produce nonredundant global edge vector and surface lists.

o PREPARATION: The vector and surface lists are operated on to produce vector and surface data arrays to speed the down stream computations. For example, the minimum/maximum values for each vector and surface in the display co-ordinate system is computed. Spatial sorting of the vector and surface data is performed. Illustrated in figure 3 is a simplistic view of this technique. Based on the complexity of the model, n X n mesh X-Y grids are imposed on the model and lists of vectors and surfaces are generated for each spatial cell via bucket sorting. Separate grids are used for the vector and surface X-Y bucket sorts. The grid density for the vector X-Y sort is based on the number of edge vectors due to surface or solid elements; for surface X-Y sorting the grid density is based on the number of surfaces. The functional relationships between these measures of model complexity and grid densities were set heuristically by varying the grid densities and observing the performance of the hidden line routine for several models. For the smallest models a grid density of 3 X 3 is used, and for the largest models a 13 X 13 grid density is used in the JONES-D hidden line routine. For models significantly more complex than the largest test models, a higher grid density

would be required for optimum performance. The lists of vectors and surfaces for each of the X-Y cells are then sorted by depth (Z). The end result of the spatial sorting are depth sorted cell vector and surface lists.

o EDGE VISIBILITY: A vector ($V_I$) is sequentially pulled from the global vector list. Its location in the spatial map, figure 3, is determined. The list of cell vectors is binary searched to find the depth to limit the search for vectors that intersect with $V_I$. Its intersection with all vectors that are ahead of it, are surface edges, and not found to be invisible by a prior calculations, are determined. A segmented vector is created from $V_I$ using its end points and the calculated points of intersection. Each segment is either all visible or invisible. The mid-point of each segment is computed and checked against the appropriate cell surface list to ascertain visibility. This requires finding the surface cell that contains the vector segment mid-point and then performing a binary search to find the depth in the cell surface list to limit the search for surfaces that hide the segment midpoint. Containment and depth tests are then performed to ascertain midpoint visibility.

o RETURN DATA: At the completion of the plot some of the data arrays created in JONES-D are passed to the main NPLOT routine to facilitate the labeling of the visible grid points and elements.

Several restrictions on the code, as implemented, should be noted but they in general have no impact on the plotting of NASTRAN finite element models.

o A line penetrating a surface results in a visible plot error. This is desirable for NASTRAN plotting since this usually indicates a modelling error.

o Grid points are required at the points where elements intersect. This normally is the case in NASTRAN models.

o Surface and solid element topologies must be reduced to four node flat surfaces. This presents no problems for commonly used NASTRAN elements. Triangles through 20 node brick elements are processed by NPLOT to this format.

## PERFORMANCE

The performance of the JONES-D hidden line technique was evaluated several different ways: on an absolute basis, in comparison with other plot types, and in comparison with other hidden line methods. In order to present an accurate picture several things should be noted:

a. The run times (CPU and elapsed) presented herein include the time to execute the plot function module (hidden line, halo line or normal plot), run the PLOT 10 module, and paint the picture on a CRT screen.

b. The CPU times show about a $\pm 5$ percent variation due to the work load on the VAX.

One set of tests consisted of running each of the 14 test models (table 1) to generate normal plots, haloed plots and hidden line plots using NPLOT, and recording the CPU and elapsed times for each of the plots. To eliminate any bias due to direction sensitivity, each run consisted of making three plots in orthogonal directions, and averaging the CPU and elapsed time. One measure of the performance of JONES-D is presented in figure 4 in the form of a plot of the edge vector processing rate (vectors per CPU second) as a function of problem size. Each data point represents the average processing rate for one of the 14 test models. The overall processing rate appears to be fairly linear with an average rate of 103.7 vectors per CPU second. The slight roll off at the high end of model size was attributed to the increase page faulting generated by these larger models. Recent experience using version 3.0 of the VAX operating system and a larger working set size (512 kbytes) increased the processing rate for these larger models by about ten percent.

Presented in table 2 is a comparison of the processing rates for normal (all lines visible) plotting, halo plotting, and hidden line plotting. The average processing rate for normal plotting was 134.7 vectors per CPU second, for hidden line, 103.7, and for haloed line, 55.7. At this point, it should be noted that the element load section of NPLOT was designed to flag certain classes of highly redundant edge vectors that result from solid elements. These vectors are eliminated very early in the JONES-D hidden line routine. Thus, these vectors were not counted in calculating the processing speed of the hidden line routine. The only two models with solid elements used in the effort, MIRROR and MIRRORH, show the effect of this procedure. For example,

the CPU time to hidden line plot MIRROR was 9.1 seconds. While MIRROR norminally consists of 1872 vectors, only 828 vectors need to be fully processed by the hidden line routine; this results in a calculated processing rate of 91 vectors per CPU second for the hidden line plotting.

The CPU time performance of the JONES-D hidden line technique for the 14 test models is compared to that for normal and haloed plots in table 3. As was expected, halo plotting was the slowest. In most cases, normal plotting was slightly faster than hidden line plotting. On an average CPU time basis, normal plotting was 16 percent faster than hidden line plotting, and halo plotting was 131 percent slower than hidden line plotting.

To the interactive user, elapsed (wall clock) time can be more significant than CPU time. Presented in table 4 is a comparison, based on elapsed time, of hidden line plotting, haloed plotting, and normal plotting for the 14 test models. In all cases except one, hidden line plotting was the quickest. In all cases haloed line plotting was the slowest. On average, the elapsed time for normal plotting was 15 percent slower than hidden line plotting, and haloed line plotting was 198 percent slower than hidden line plotting.

The JONES-D algorithm was compared to five other hidden line methods. Three of the comparisons were based on hands-on use of the alternative algorithms and two of the comparisons were based on published performance data. This was not meant to be a precise, exhaustive, and detailed comparative study but rather to illustrate relatively large differences in performance when dealing with the target application, which was NASTRAN finite element model plotting.

a. The Watkins hidden line/surface method is used in the MOVIE program. A VAX implementation of MOVIE was used for this study. The Watkins method does not support line element types so an all surface model, the surface elements of the FLFSS model, was used for this comparison. In this form the test model contained 857 surfaces. The CPU time for MOVIE was 41.5 seconds for generating the hidden line plot; the corresponding time for JONES-D was 11.6 CPU seconds. The MOVIE plot contained a few plot errors; however, the author's understanding is that the most recent version of MOVIE is fixed in this regard but that it runs a little slower.

b. The SKETCH hidden line routine recently developed by Hedgley (reference 12) was obtained and converted to the VAX 11/780 computer. The routine as delivered was limited to about 250 polygons, therefore, a relatively small model was selected for testing (the CQUADS of the FSS model, 183 surfaces). The CPU time for SKETCH was 19.3 seconds for the hidden line plot; the corresponding time for JONES-D was 3.5 CPU seconds. This level of performance for SKETCH ($\sim$9 polygons per second) seems consistent with the data presented in reference 12. In that reference the processing rate for SKETCH on a CDC 6500 computer (slightly faster than a VAX 11/780) for finite element type models was about 10 polygons per CPU second. Some plot errors were observed in the SKETCH generated plot, but this may be due to the fact that single precision (32 bit) data formats were used on the VAX; whereas, the original implementation used a 60 bit format (CDC 6500).

c. The VIEW thermal view factor program developed for NASA/LaRC by Professor Emery of the University of Washington contains a hidden line routine. The NCONES thermal model, figure 5, was used for comparison testing. The CPU time for VIEW was 48.8 CPU seconds for the hidden line plot, the corresponding time for JONES-D was 5.05 CPU seconds. VIEW was developed on a computer (PDP 11/45) with limited memory resources, so by necessity it does a lot of FORTRAN I/O which may account for its relatively slow performance. VIEW does seem very reliable, and its plots have contained no obvious plot errors.

d. A fast hidden line method was developed by Bareau (reference 2) for the plotting of finite element models. We were not able to obtain the code for this technique, so no hands-on testing was performed. Based on the data presented in the referenced paper it seems quite fast. For example, on a CDC 6500 computer, (about 1.2 times faster than a VAX 11/780), its performance was about 55 edge vectors per CPU second. This level of performance is 2.5 to 3.0 times slower than JONES-D.

e. In a recent paper, (reference 13) Wittram presented a new fast hidden line method. Again we were not able to obtain the code necessary for hands-on testing. For a scene with 1029 faces, the CPU time is given as 20 CPU seconds; to this must be added the X-sort time of about 15 CPU seconds for a total CPU time of 35 seconds on a ICL 1906S computer. This corresponds to a processing rate of about 30 surfaces per CPU second; the performance of JONES-D for a similar size model (FLFSS, surface elements only) is about 74

surfaces per CPU second. Unfortunately, the relative performance of the ICL 1906S to the VAX 11/780 is somewhat uncertain. Based on Wittram's comment that a ICL 1906S is about 10 times slower than an IBM 3081, one could surmise that a VAX 11/780 is about 20 percent faster than an ICL 1906S. If this is true, then Wittram's method is about two times slower than JONES-D.

## CONCLUSIONS

A high speed hidden line technique has been developed to facilitate the plotting of NASTRAN finite element models. Based on testing using many (14) different models, the new hidden line algorithm (JONES-D) appears to be:

a. Very Fast: Its speed equals that for normal (all lines visible) plotting, and when compared to other existing methods, it appears to be substantially faster.

b. Very Reliable: No plot errors have been observed using the new method to plot NASTRAN models.

The new algorithm has been made part of the NPLOT NASTRAN plot package and has been used at the GSFC by structural analysts for normal production tasks. It is interesting to note that analysts when given a choice, seldom use normal, all lines visible, plotting.

## REFERENCES


1. Newman, W.M., and Sproull, R.F., Principles of Interactive Computer Graphics, McGraw-Hill Co., 1979.

2. Bareau, M., "Convenient Representation Method For Spatial Finite Element Structures", Computers & Structures, 10(5), October 1979, pp. 815-819.

3. Appel, A., Rohlf, F.J., Stein, A.J., "The Haloed Line Effect for Hidden Line Elimination", Computer Graphics, 13(3), August 1979, pp. 151-157.

4. Franklin, W.R., "A Linear Time Exact Hidden Surface Algorithm", Computer Graphics, 14(3), August 1980, pp. 117-123.

5. Giloi, W.K., Interactive Computer Graphics, Data Structures, Algorithms, Languages, Prentice-Hall Inc., 1978.

6. Griffiths, J.G., "A Surface Display Algorithm", Computer Aided Design, 10(1), January 1978, pp. 65-73.

7. Griffiths, J.G., "A Bibliography of Hidden-Line and Hidden-Surface Algorithms", Computer Aided Design, 10(3), May 1978, pp. 203-206.

8. Griffiths, J.G., "Tape-Oriented Hidden Line Algorithm", Computer Aided Design, 13 (1), January 1981, pp. 19-26.

9. Watkins, G.S., A Real Time Visible Surface Algorithm, University of Utah, UTEC-CSc-70 101, June 1970.

10. Emery, A.F. VIEW, Department of Mechanical Engineering University of Washington, 1982.

11. Foley, J.D., and Van Dam, A., Fundamentals of Computer Graphics, Addision-Wesley, 1982.

12. Hedgley, D.R. Jr., A General Solution to the Hidden Line Problem, NASA RP-1085, March 1982.

13. Brittram, M., "Hidden-Line Algorithm for Scenes of High Complexity", Computer Aided Design, 13(4), July 1981, pp. 187-192.

| MODEL | GRID POINTS | LINE ELEMENTS | SURFACE ELEMENTS | SOLID ELEMENTS |
|---|---|---|---|---|
| JPL | 55 | 9 | 67 | 0 |
| FSS | 167 | 352 | 279 | 0 |
| CDH | 210 | 180 | 191 | 0 |
| MAYPOLE | 231 | 542 | 96 | 0 |
| OSSFF | 315 | 462 | 130 | 0 |
| CUBE8D | 408 | 382 | 342 | 0 |
| BBXRT | 535 | 0 | 576 | 0 |
| FLFSS | 604 | 1235 | 857 | 0 |
| MMS | 678 | 678 | 519 | 0 |
| DHRS | 703 | 717 | 779 | 0 |
| MIRROR | 540 | 0 | 180 | 720 |
| FEM | 1268 | 1146 | 1012 | 0 |
| LSDBUCK | 1502 | 1683 | 1382 | 0 |
| MIRRORH | 1183 | 0 | 0 | 864 |

NASTRAN Model Data

Table 1

| MODEL | VECTORS | SPEED VECTOR/CPU SEC. NORMAL | HALO | HIDDEN |
|---|---|---|---|---|
| JPL | 126 | 131.3 | 68.1 | 97.6 |
| FSS | 402 | 128.8 | 58.6 | 106.1 |
| CDH | 472 | 135.2 | 81.4 | 131.5 |
| MAYPOLE | 518 | 136.3 | 15.4 | 78.4 |
| OSSFF | 525 | 127.4 | 65.1 | 147.5 |
| COBE8D | 894 | 134.4 | 59.8 | 124.7 |
| BBXRT | 1084 | 139.0 | 60.5 | 97.7 |
| FLFSS | 1375 | 133.5 | 55.1 | 98.4 |
| MMS | 1430 | 137.5 | 62.6 | 100.3 |
| DHRS | 1561 | 136.8 | 53.9 | 102 |
| MIRROR | 1872/828* | 133.1 | 31.7 | 91.0 |
| FEM | 2413 | 136.6 | 46.1 | 88.1 |
| LSDBUCK | 3056 | 141.1 | 69.5 | 82.7 |
| MIRRORH | 3198/1152* | 135.2 | 52.1 | 106.5 |

* Number of vector for hidden line processing

Vector Processing Speed

Table 2

| MODEL | VECTORS | CPU TIME (SEC.) NORMAL | HALO | HIDDEN |
|---|---|---|---|---|
| JPL | 126 | .96 | 1.85 | 1.29 |
| FSS | 402 | 3.12 | 6.86 | 3.79 |
| CDH | 472 | 3.49 | 5.80 | 3.59 |
| MAYPOLE | 518 | 3.80 | 33.59 | 6.61 |
| OSSFF | 525 | 4.12 | 8.07 | 3.56 |
| COBE8D | 894 | 6.65 | 14.96 | 7.17 |
| BBXRT | 1084 | 7.80 | 17.91 | 11.10 |
| FLFSS | 1375 | 10.30 | 24.93 | 13.97 |
| MMS | 1430 | 10.40 | 22.86 | 14.25 |
| DHRS | 1561 | 11.41 | 28.96 | 15.31 |
| MIRROR | 1872/828* | 14.06 | 58.96 | 9.10 |
| FEM | 2413 | 17.66 | 52.33 | 27.39 |
| LSDBUCK | 3056 | 21.66 | 43.99 | 36.96 |
| MIRRORH | 3198/1152* | 23.65 | 61.35 | 10.82 |

* Number of vectors for hidden line processing

CPU Time Comparison
Hidden vs. Halo vs. Normal

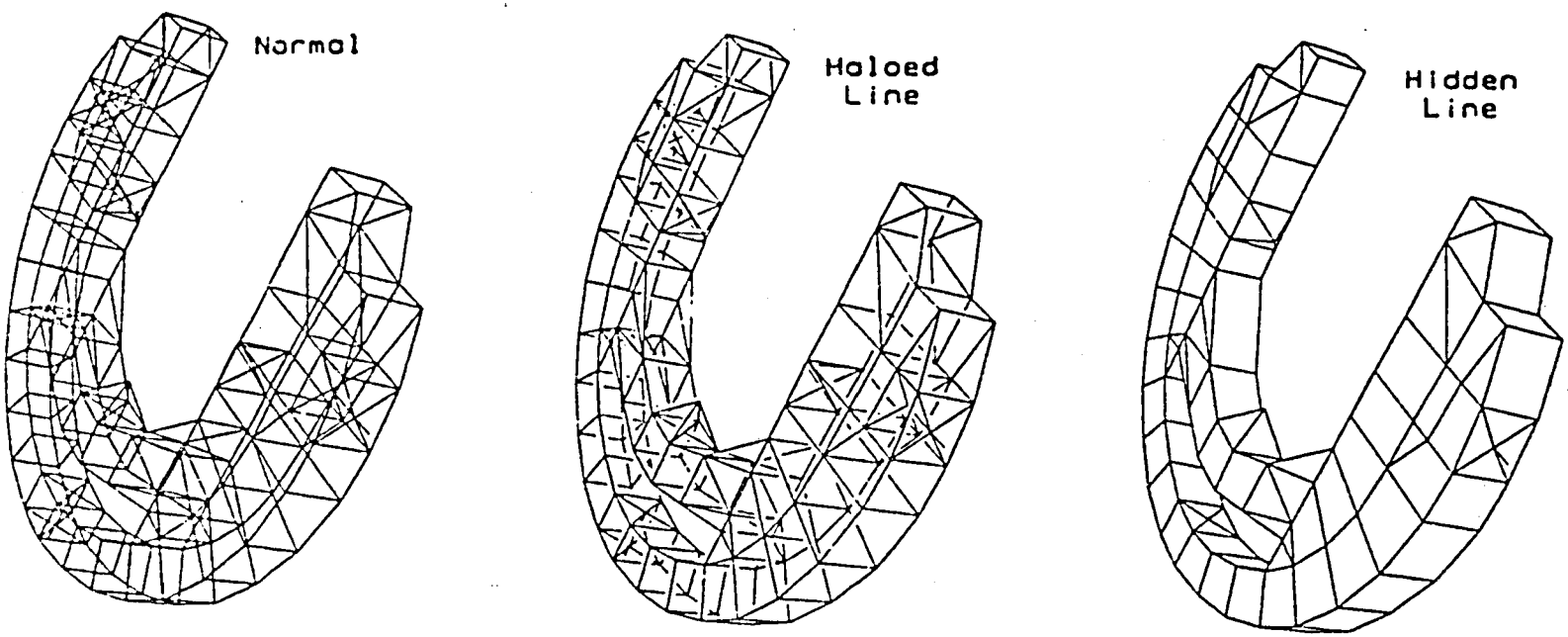
Table 3

| MODEL | VECTORS | ELAPSED TIME (SEC.) NORMAL | HALO | HIDDEN |
|---|---|---|---|---|
| JPL | 126 | 3.6 | 5.3 | 3.2 |
| FSS | 402 | 11.2 | 23.9 | 8.0 |
| CDH | 472 | 12.5 | 19.7 | 7.8 |
| MAYPOLE | 518 | 13.3 | 84.6 | 17.4 |
| OSSFF | 525 | 13.9 | 22.6 | 8.9 |
| COBE8D | 894 | 25.2 | 37.5 | 14.8 |
| BBXRT | 1084 | 27.1 | 52.1 | 23.5 |
| FLFSS | 1375 | 35.8 | 65.4 | 25.4 |
| MMS | 1430 | 34.2 | 59.9 | 27.5 |
| DHRS | 1561 | 40.0 | 70.2 | 26.9 |
| MIRROR | 1872/828* | 47.9 | 214.8 | 19.0 |
| FEM | 2413 | 61.0 | 124.9 | 60.0 |
| LSDBUCK | 3056 | 74.2 | 86.6 | 68.7 |
| MIRRORH | 3198/1152* | 85.5 | 131.6 | 23.6 |

* Number of vectors for hidden line processing

Elapsed Time Comparison
Hidden vs. Halo vs. Normal

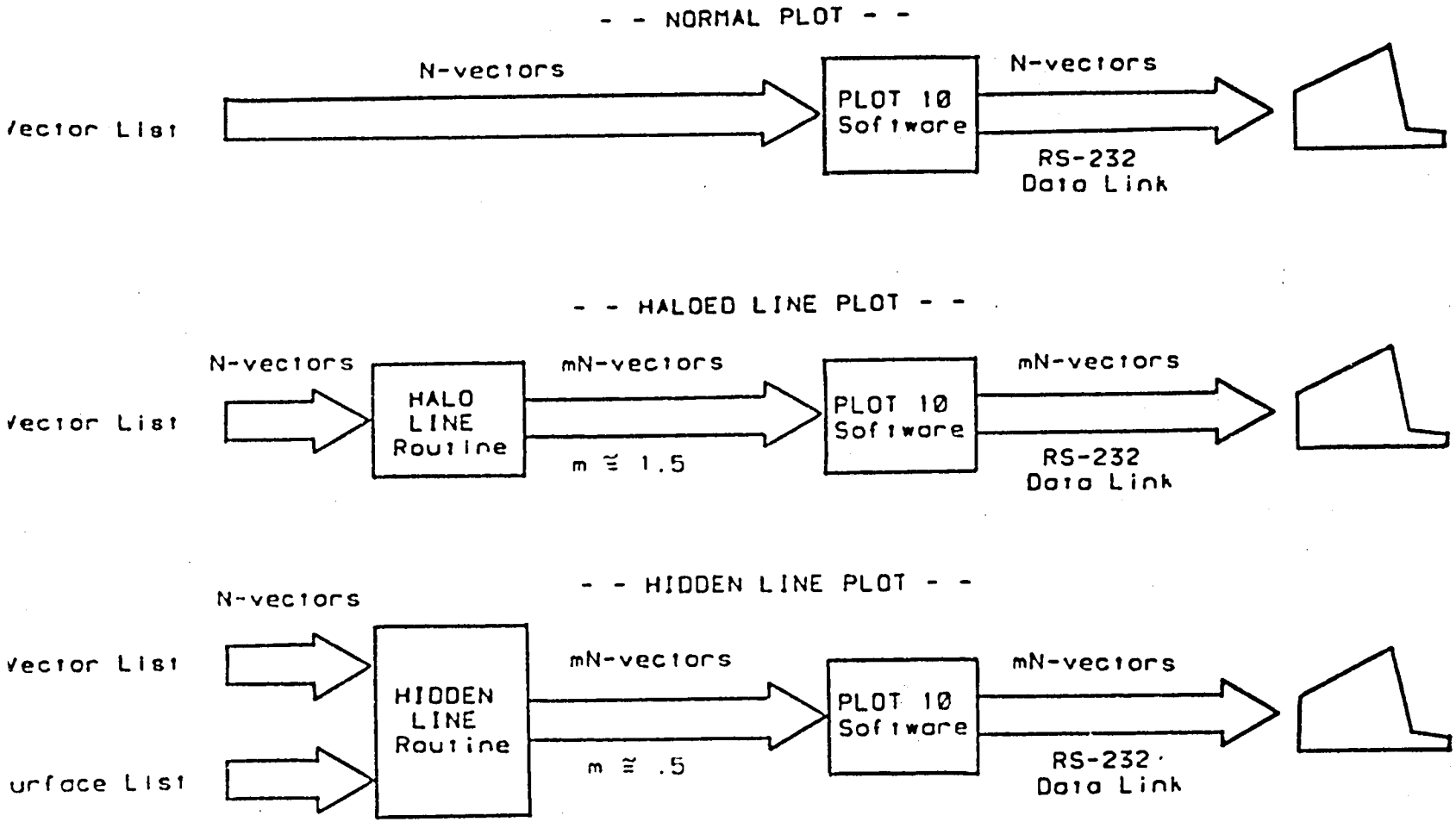
Table 4





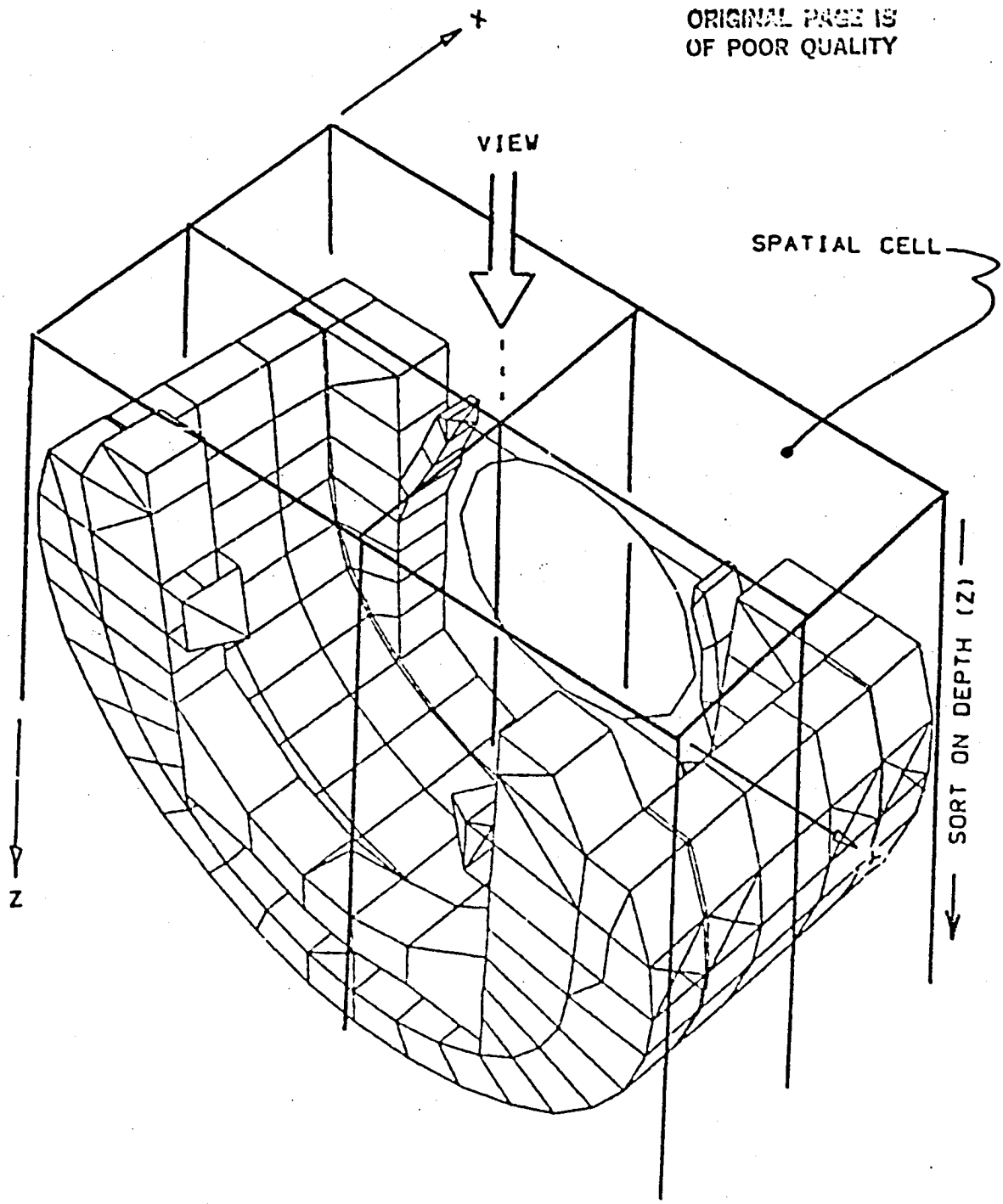

COMPARISON OF PLOT TYPES

FIGURE 1

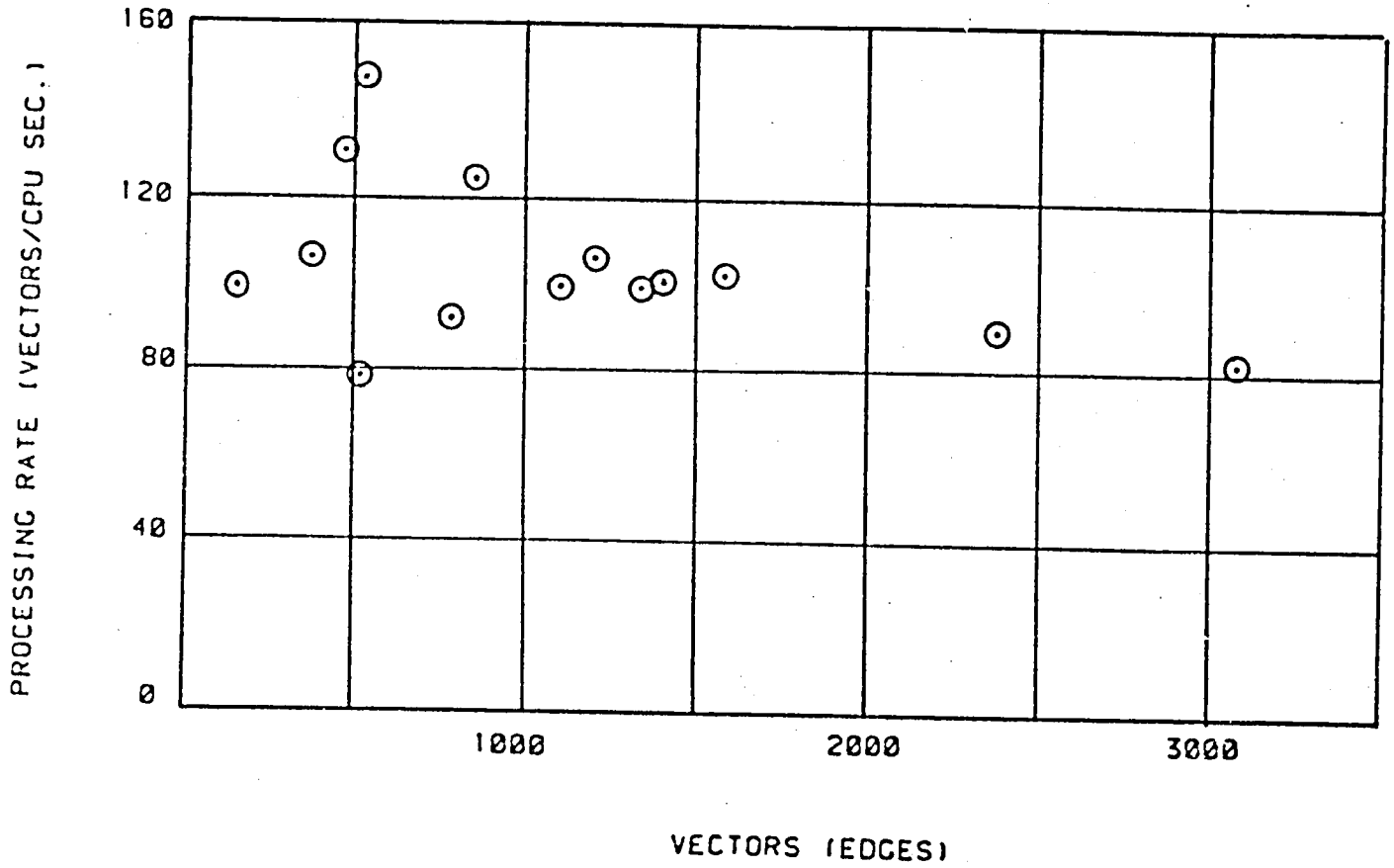



COMPARISON OF THE DATA FLOWS FOR THE DIFFERENT PLOT TYPES

FIGURE 2

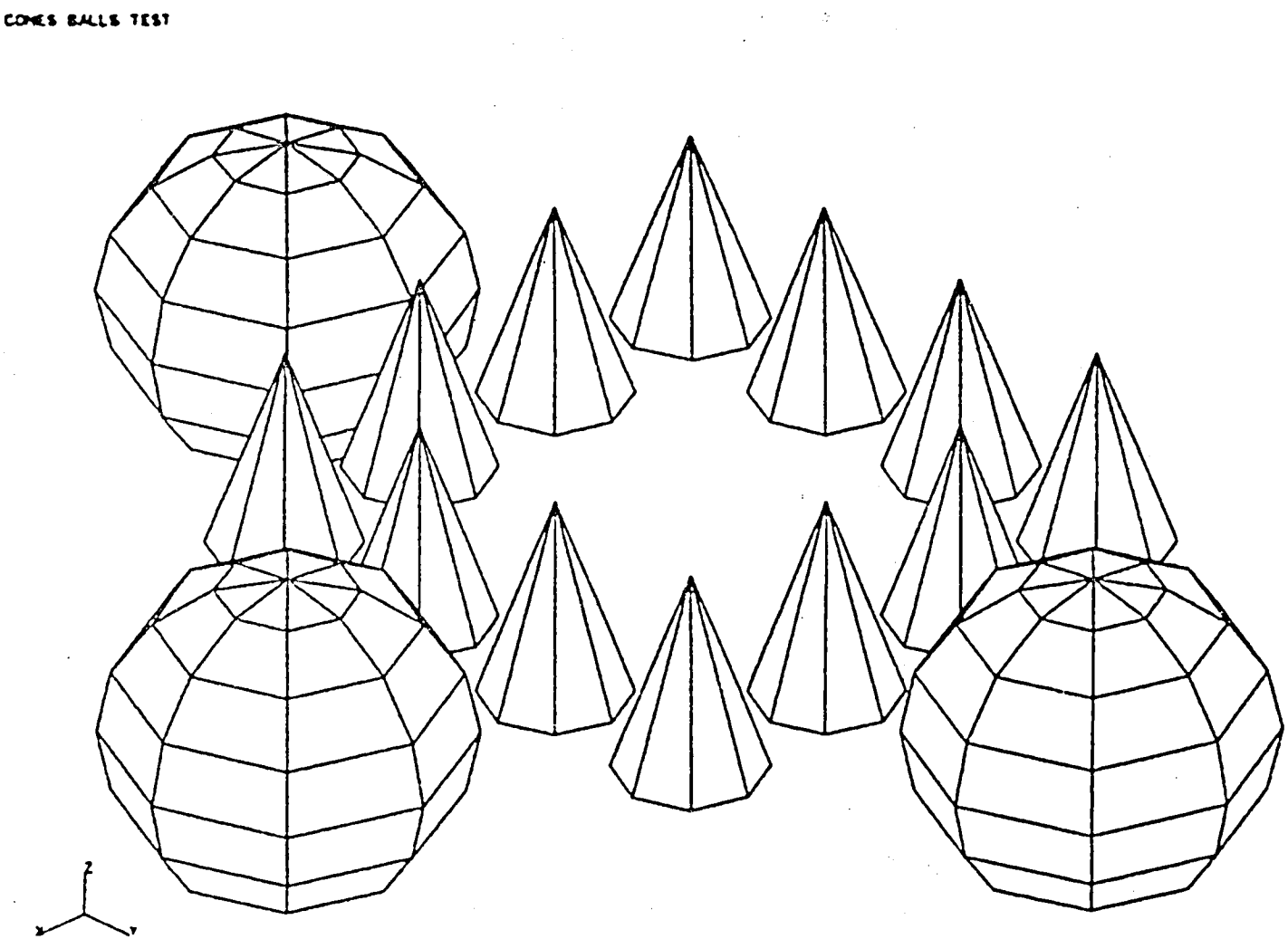



SPATIAL CELLS USED IN X-Y BUCKET SORTING




HIDDEN LINE PROCESSING SPEED

FIGURE 4



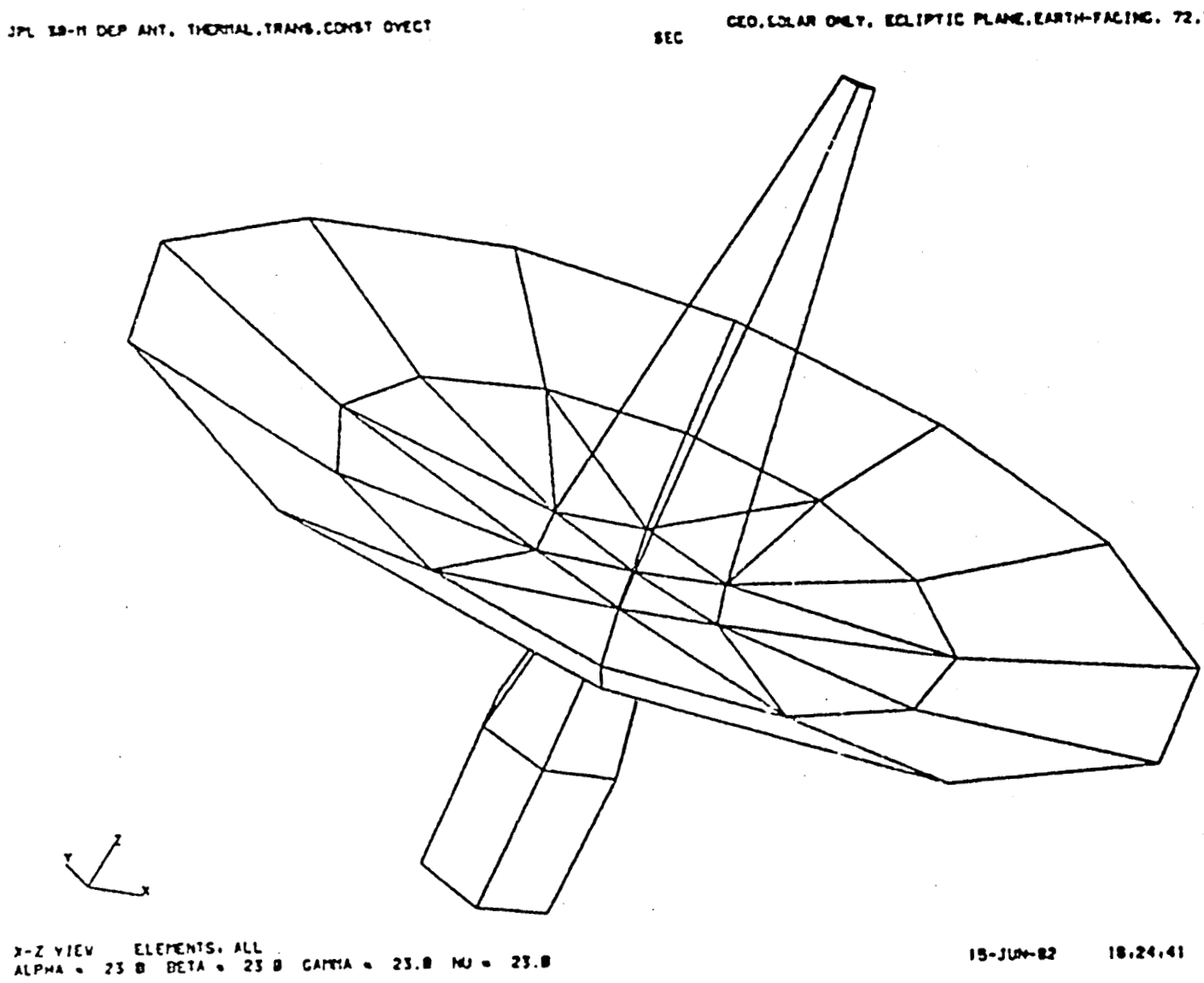

NCONES THERMAL MODEL

FIGURE 5

# APPENDIX A

## HIDDEN LINE PLOTS

Presented in this appendix are the hidden line plots for the fourteen test models (figures 1A through 14A) generated by use of the JONES-D hidden line routine and the NPLOT NASTRAN plot package. Noted on each plot is the CPU time required to generate it. Illustrated in figure 15A is the effect of element labeling and grid labeling on a hidden line plot. On complex plots, the zoom feature is essential to declutter the displayed image; presented in figure 16A is such a zoom view of the image shown in figure 15A.

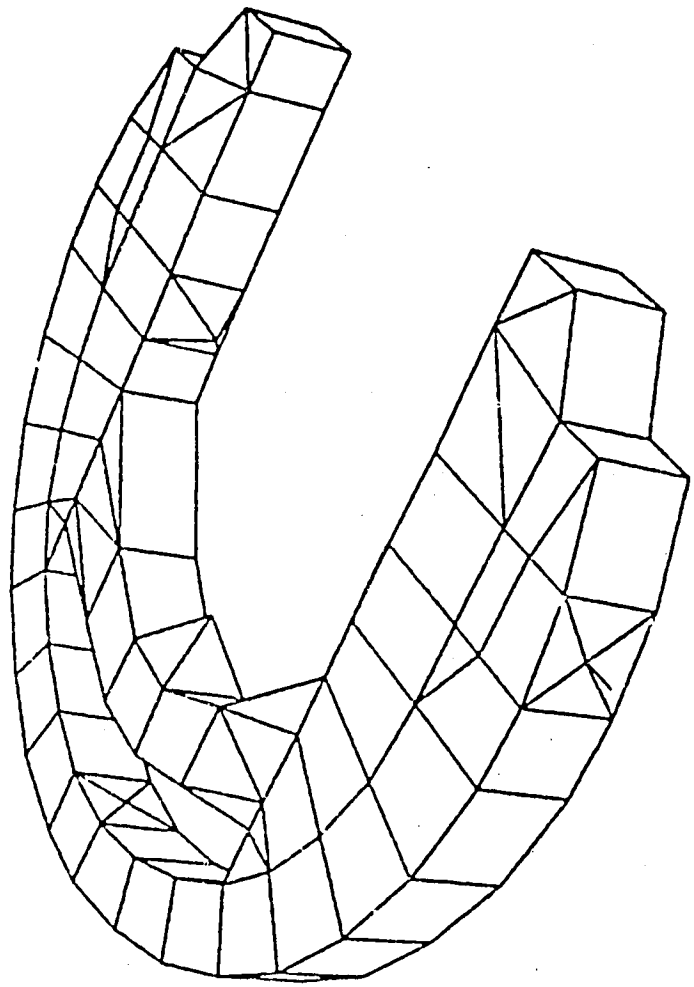





JPL MODEL ( 1.28 cpu sec. )

FIGURE 1A

CRADLE A FORWARD - LOWER POSITION ( Z = 100.0 )

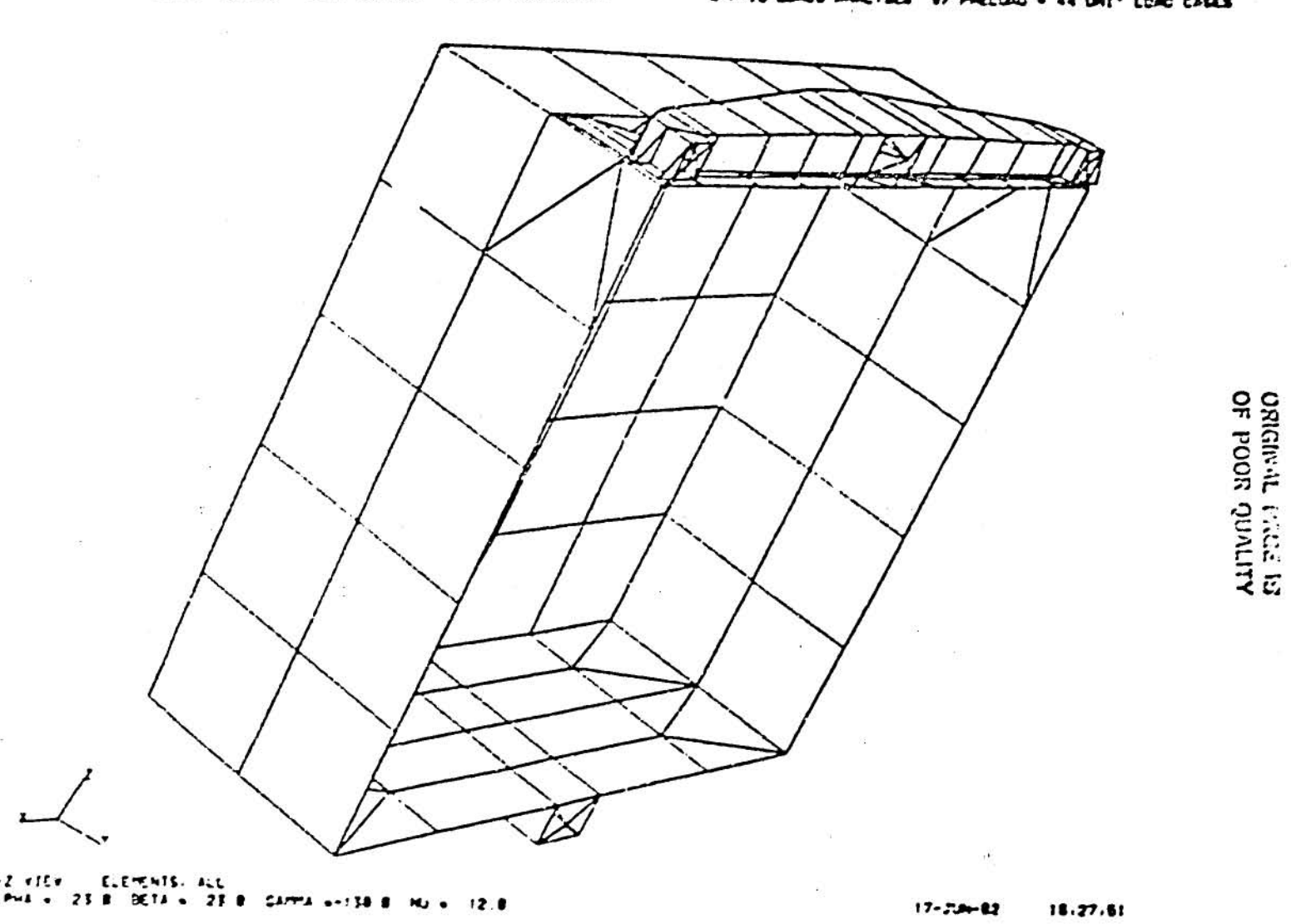


X-Z VIEW ELEMENTS: ALL
ALPHA = 23.0 BETA = 23.0 GAMMA = 23.0 MU = 23.0

15-JUN-82 18:28:37

FSS MODEL ( 3.79 cpu sec. )

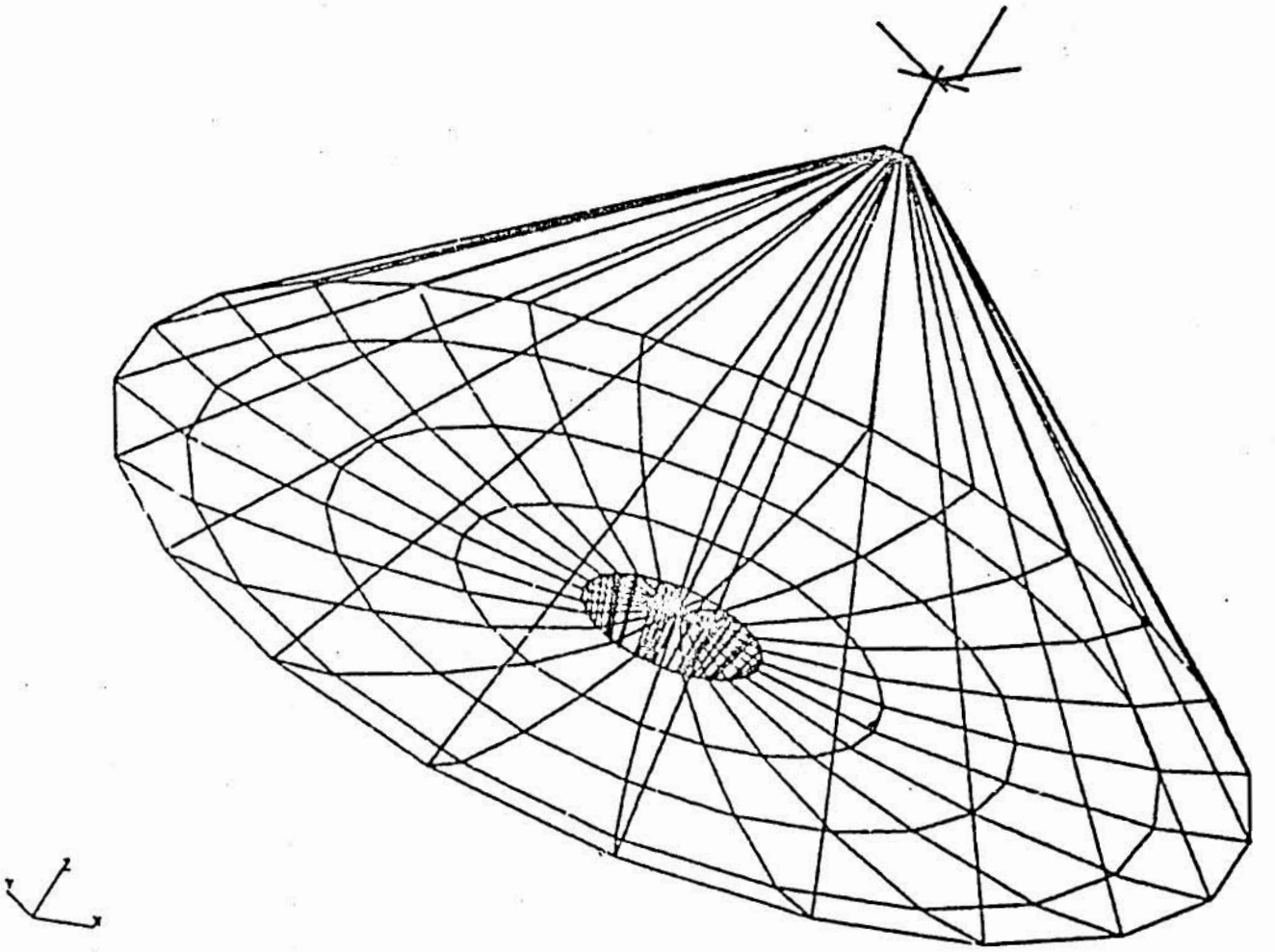
FIGURE 2A






CDH MODEL : 3 59 cpu sec. )

FIGURE 3A

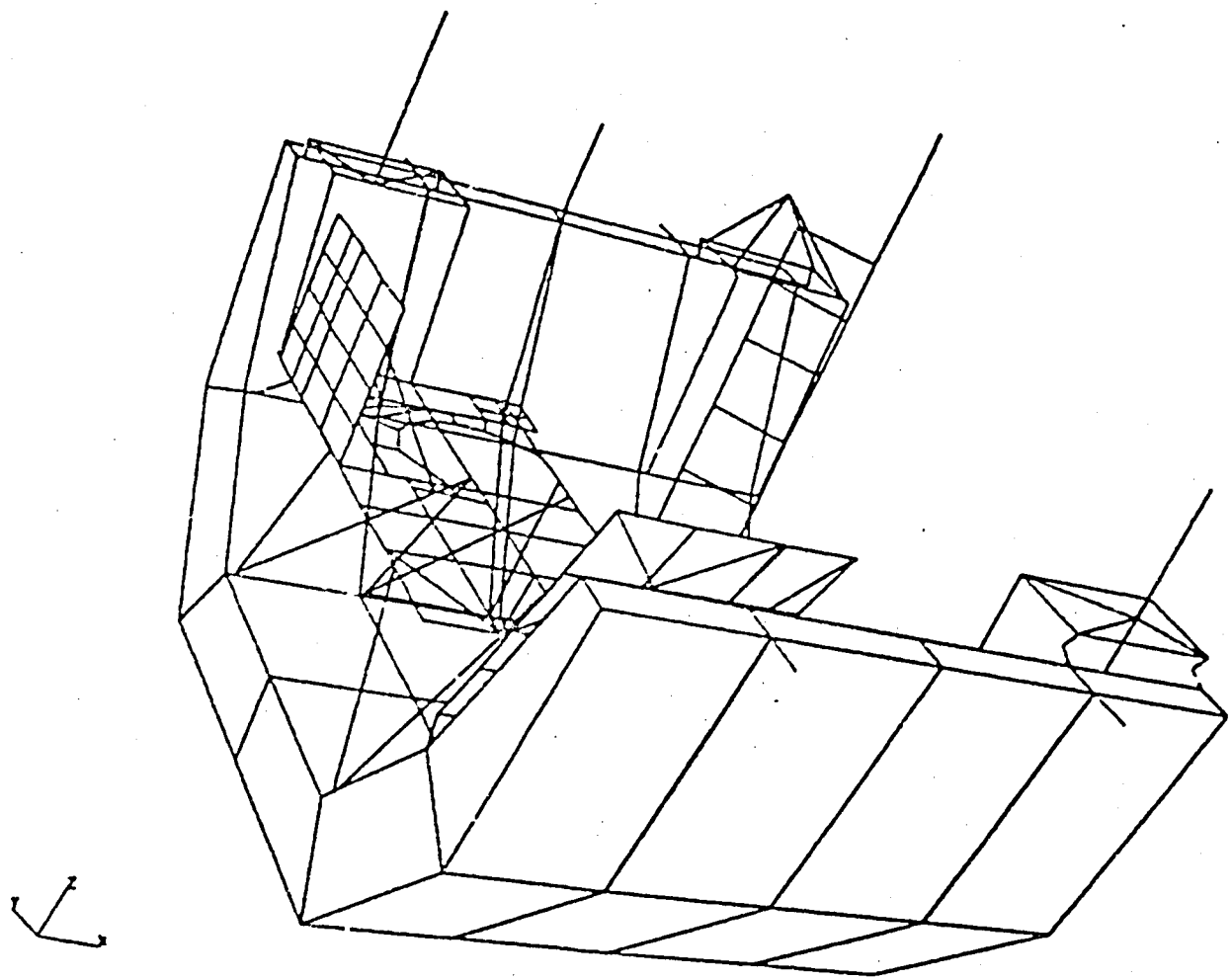



MAYPOLE MODEL ( 6.61 cpu sec. )

FIGURE 4A

OSS-1 COUPLED PALLET-PAYLOAD MODEL

LATEST PRE-MODAL SURVEY VERSION WITH SUNY REVISIONS



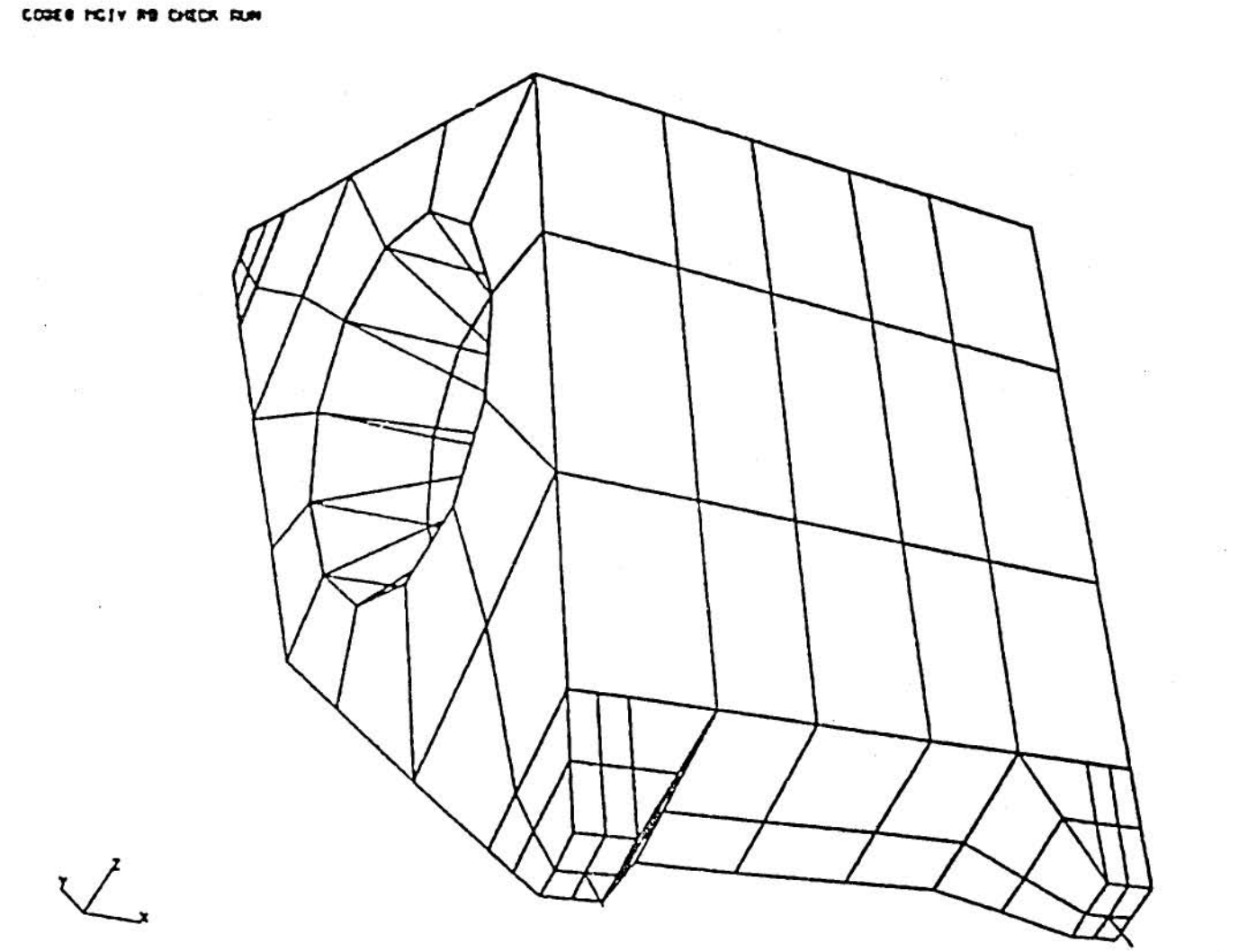
X-Z VIEW ELEMENTS: ALL
ALPHA = 23.0 BETA = 23.0 GAMMA = 23.0 MU = 23.0

17-JUN-92 16:33:20



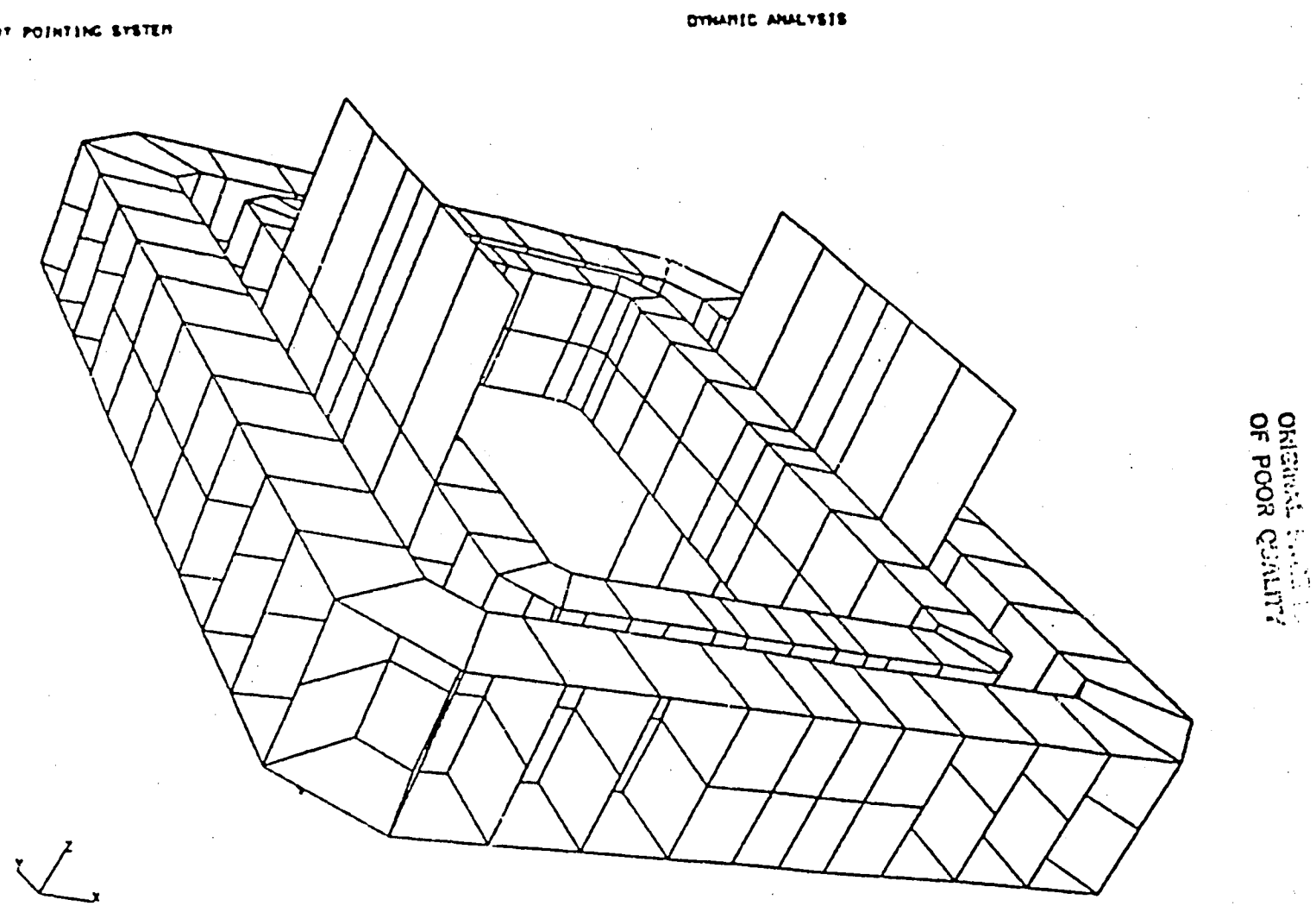
OSSFF MODEL ( 3.56 cpu sec. )

FIGURE 5A




COBE8D MODEL ( 7.17 cpu sec.)

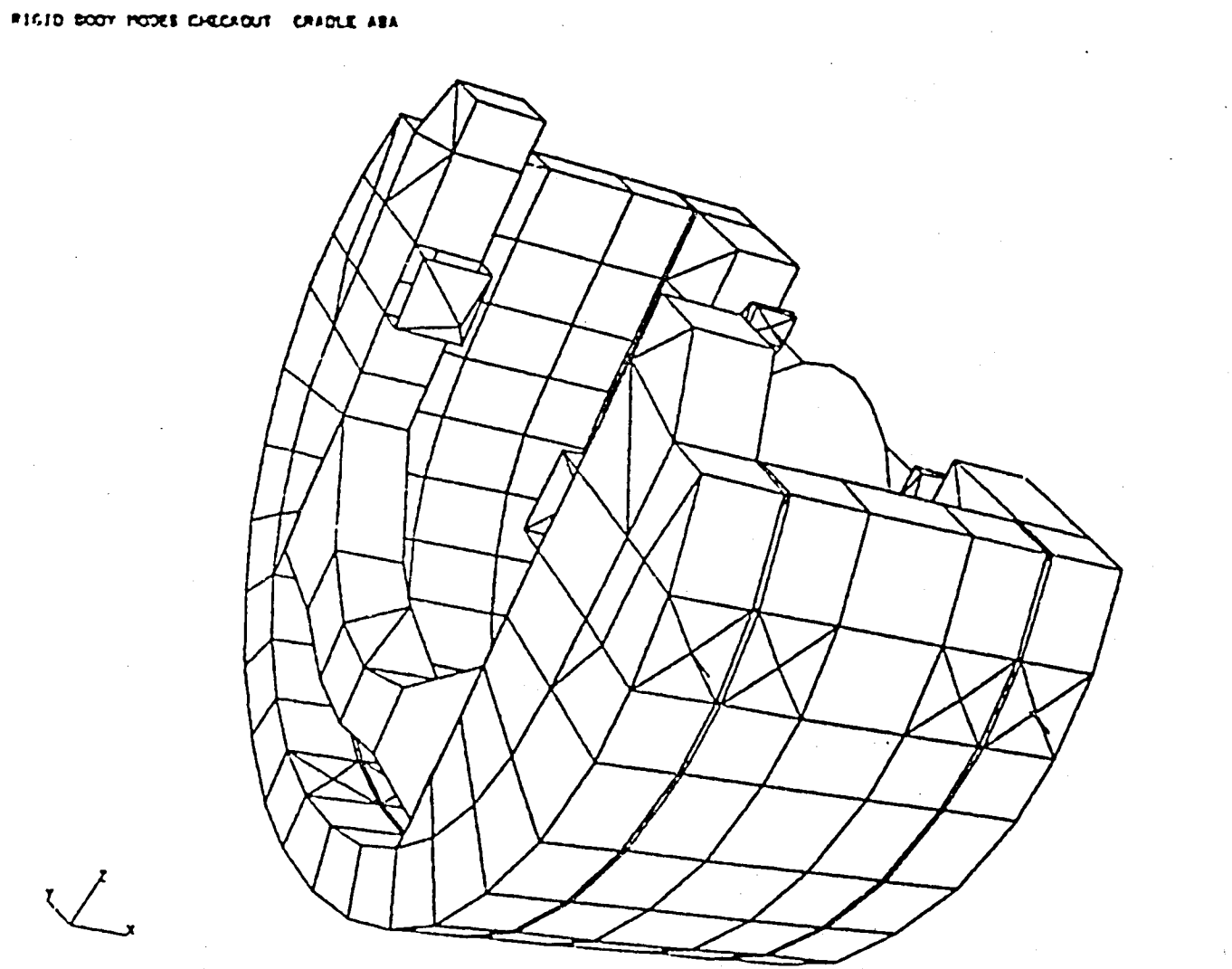
FIGURE 6A






BBXRT MODEL ( 11.10 cpu sec. )

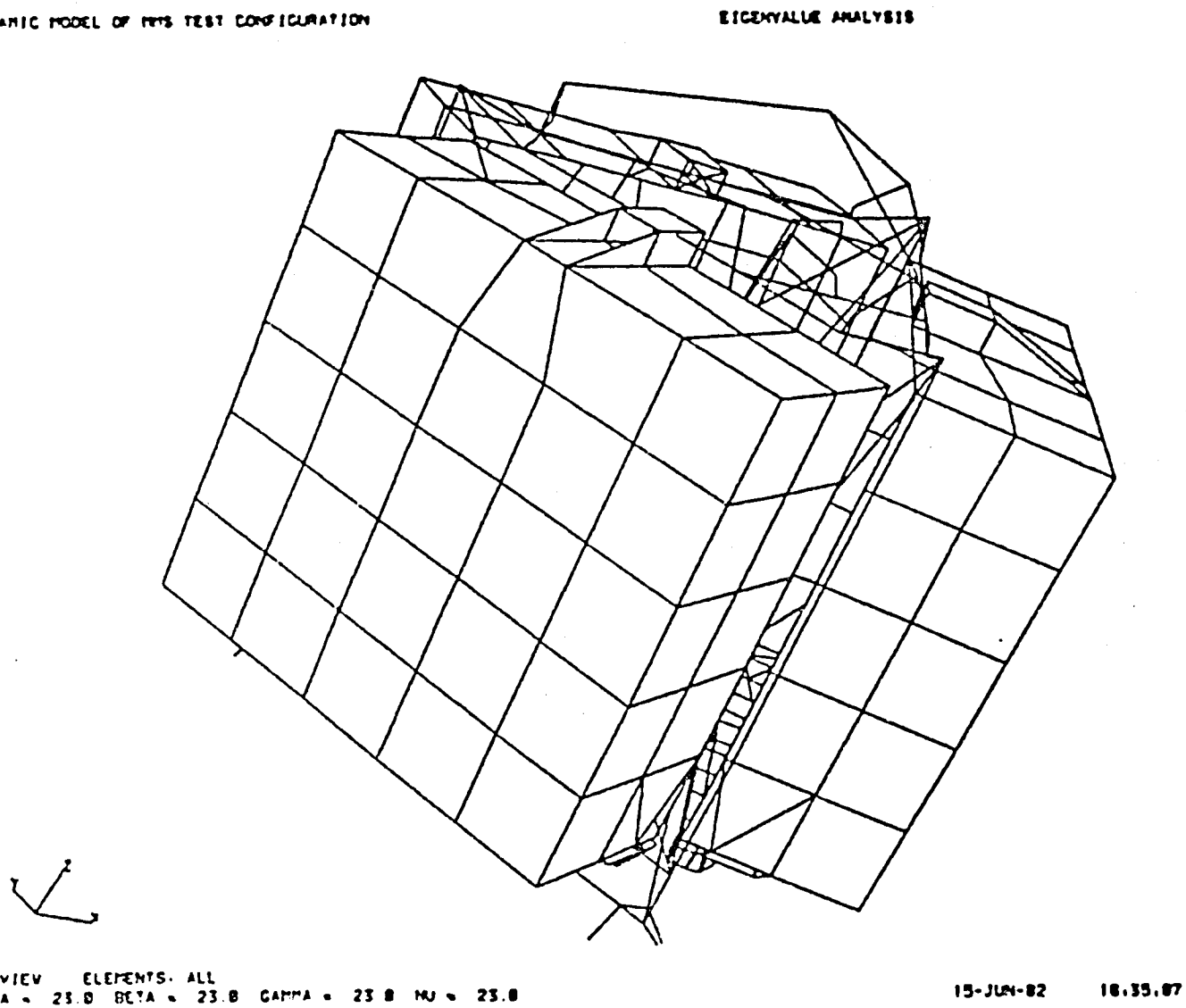
FIGURE 7A





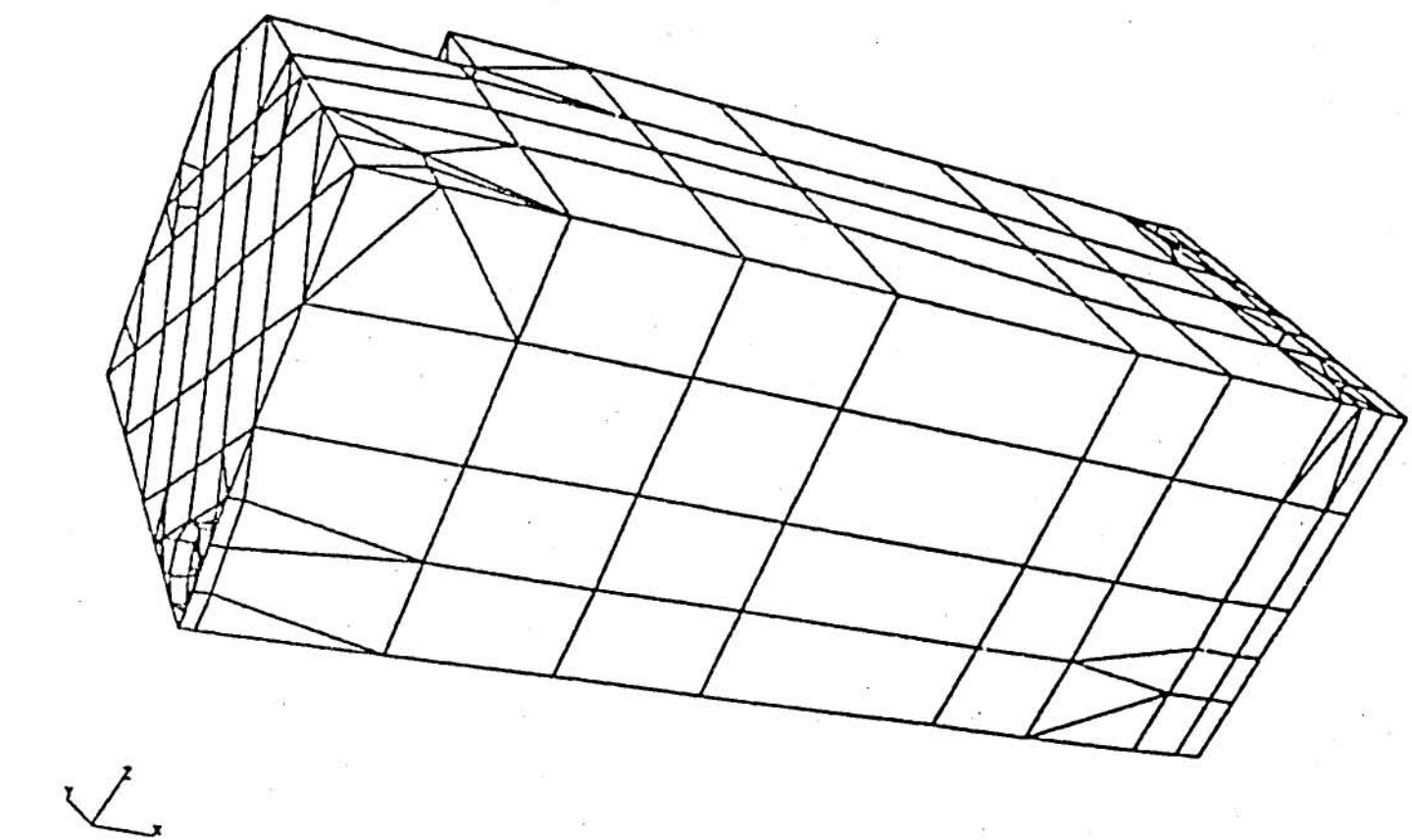

FLFSS MODEL ( 13.97 cpu sec. )

FIGURE 8A



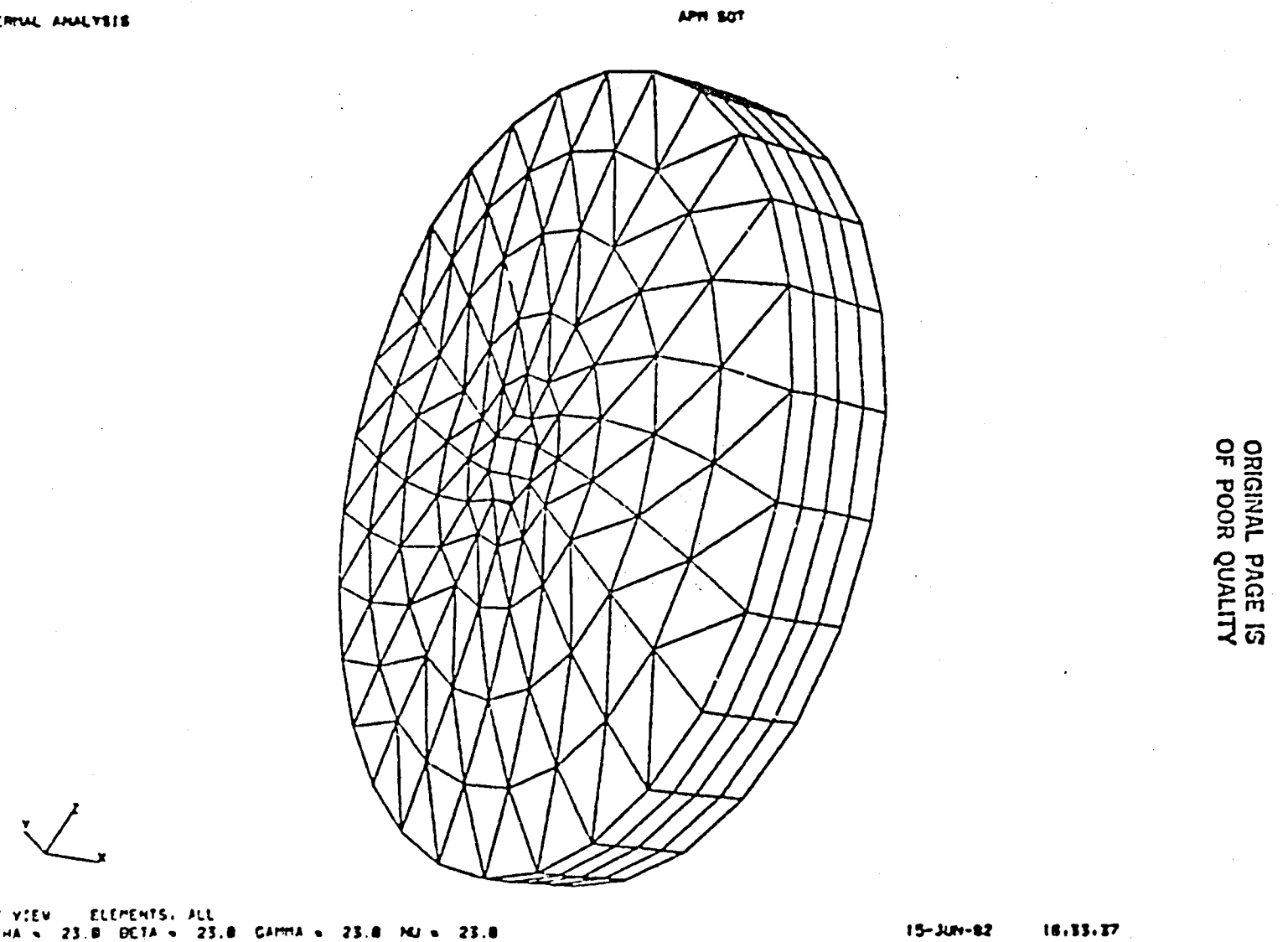

MMS MODEL ( 14.25 cpu sec. )

FIGURE 9A




DHRS MODEL ( 15.31 cpu sec. )

FIGURE 10A

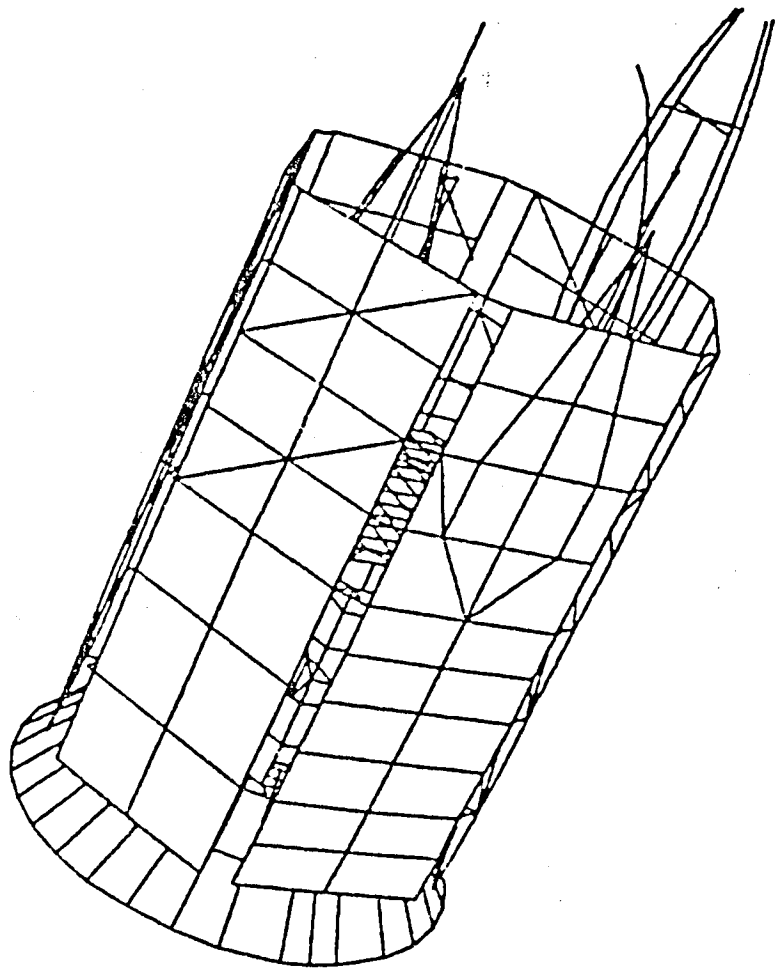



MIRROR MODEL ( 8.18 cpu sec. )

FIGURE 11A

TORSS FINITE ELEMENT MODEL

CONVERTED FROM TRIGAP (1/82)



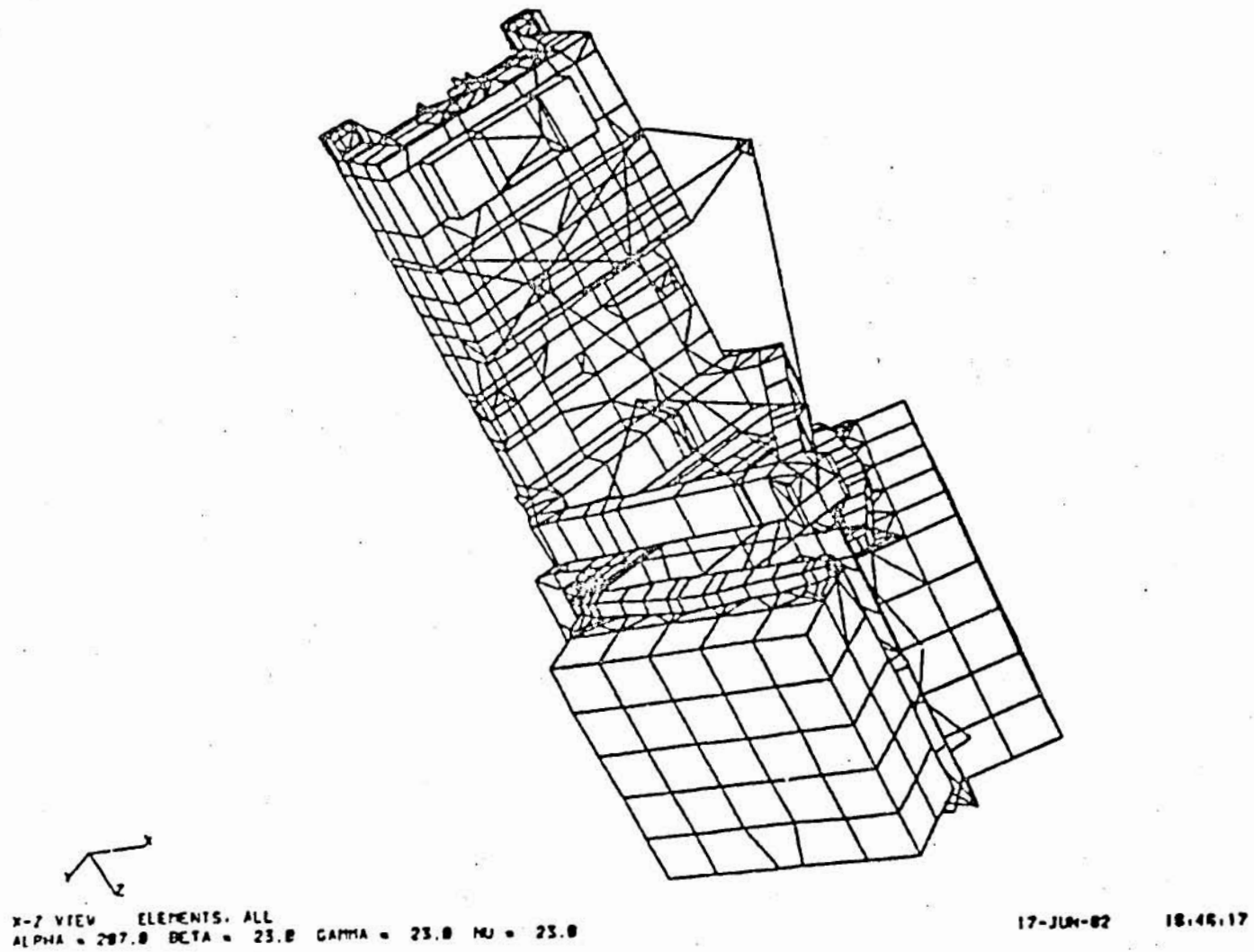
X-Z VIEW ELEMENTS: ALL
ALPHA = 23.8 BETA = 23.8 GAMMA = 23.8 MU = 23.8

17-JUN-82 16:39:24



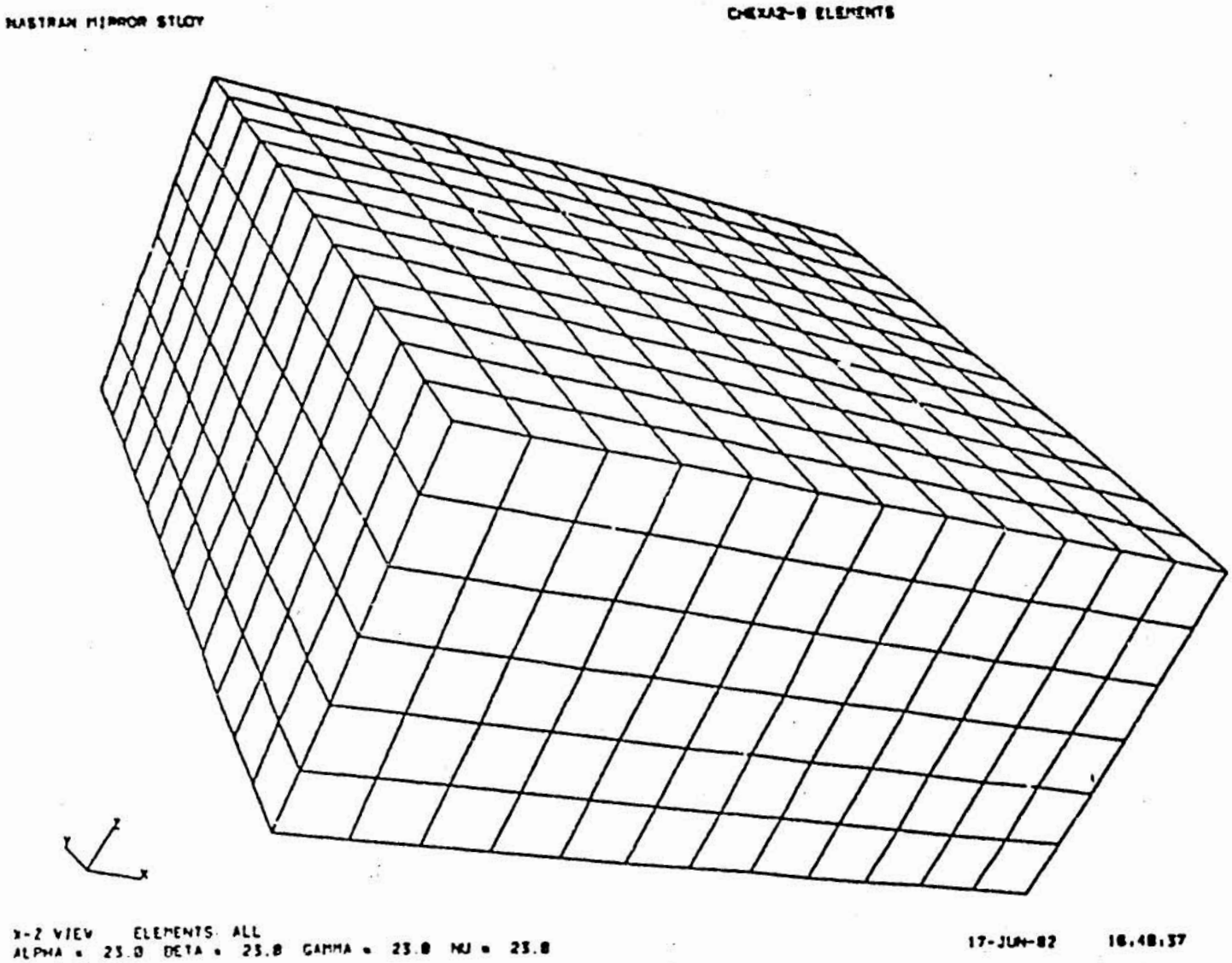
FEM MODEL ( 27.39 cpu sec. )

FIGURE 12A



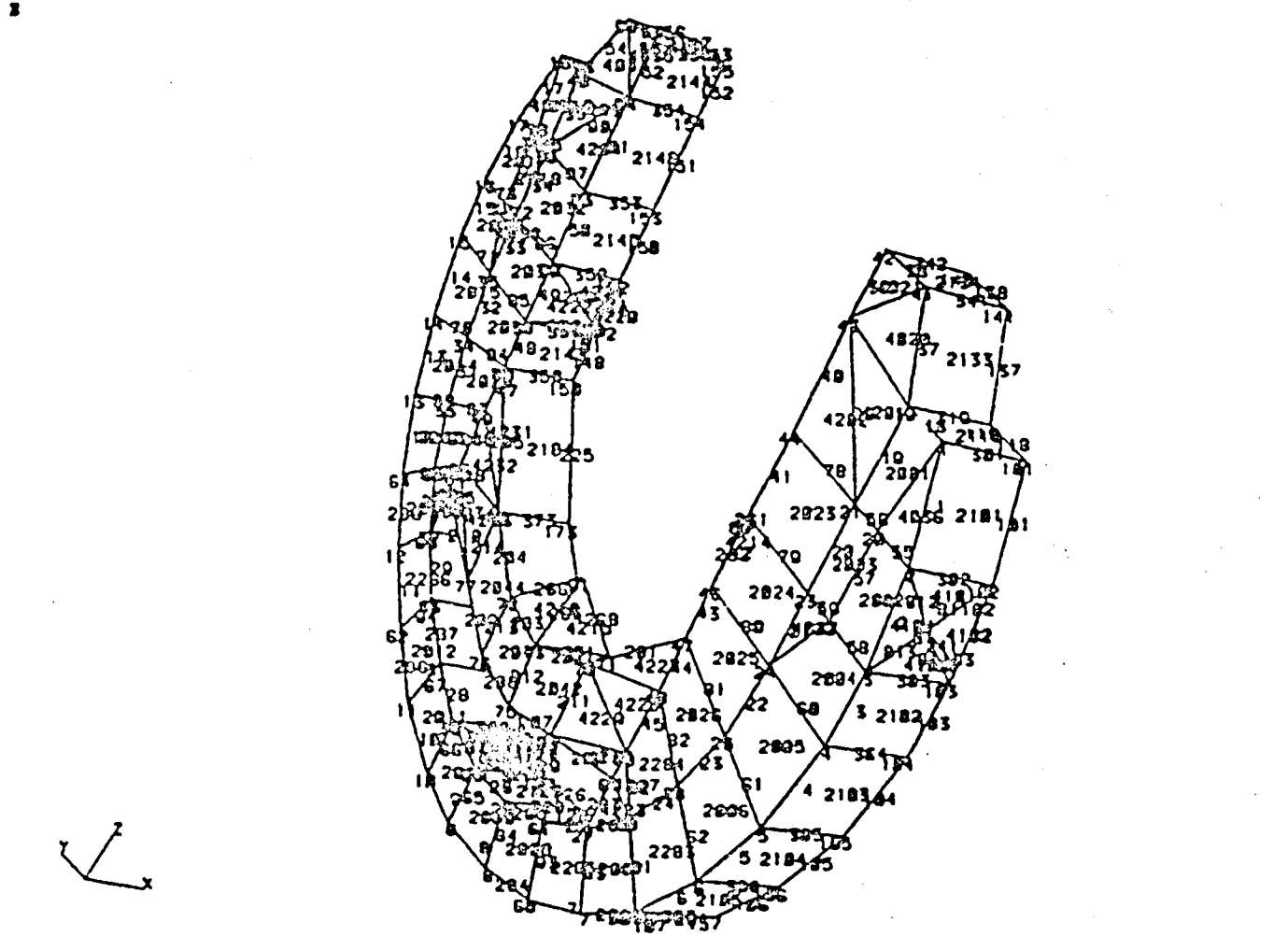

LSDBUCK MODEL ( 36.96 cpu sec. )

FIGURE 13A



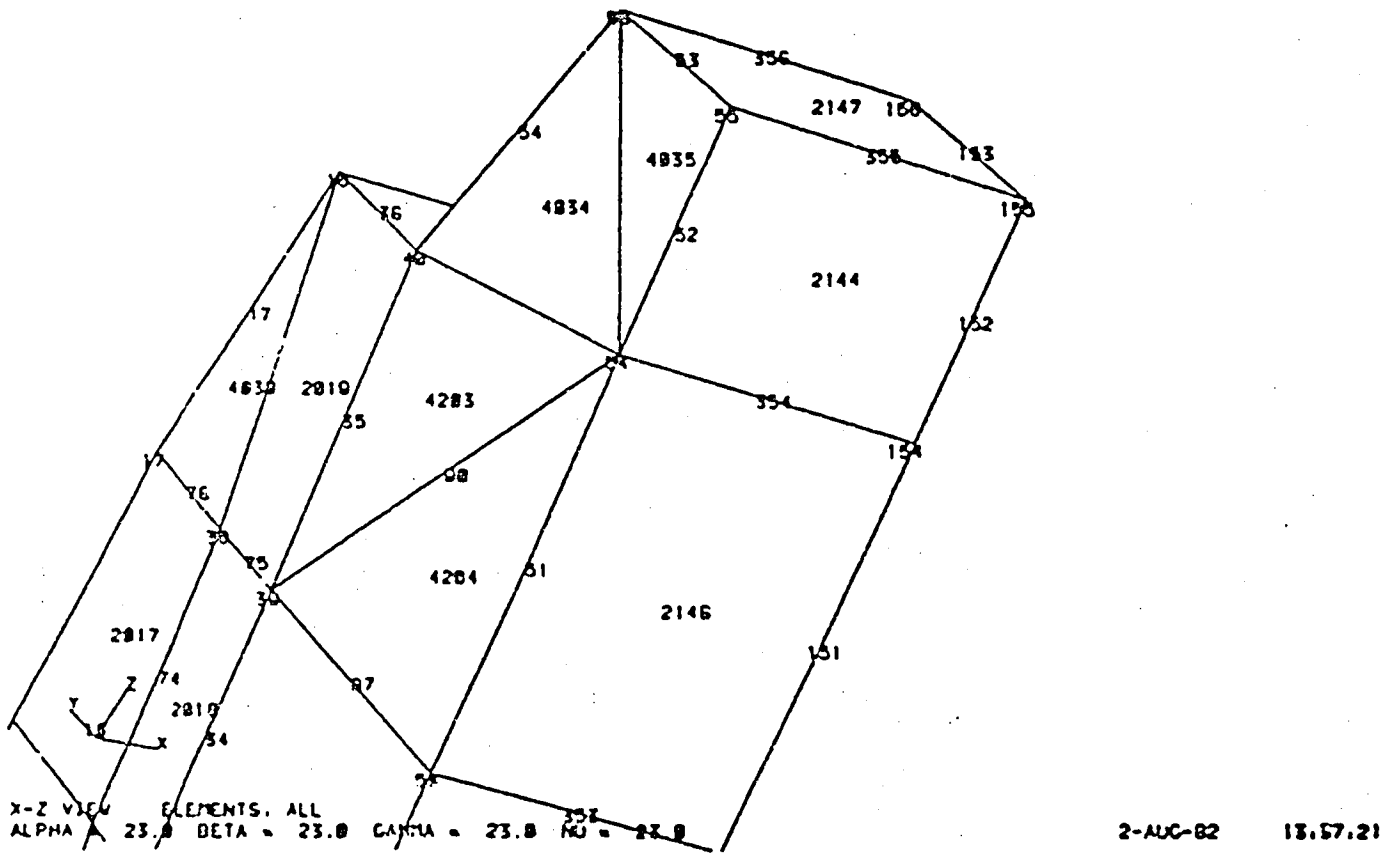

MIRRORH MODEL ( 10.82 cpu sec. )

FIGURE 14A




FSS MODEL WITH GRID AND ELEMENT LABELS

FIGURE 15A




ZOOMED PLOT OF FSS MODEL WITH GRID AND ELEMENT LABELS

FIGURE 16A

# APPENDIX B

## SOURCE LISTING

## JONES-D HIDDEN LINE ROUTINE

Presented in this appendix are the source listings of:

a. The JONES-D hidden line routine, pages B2-B11.

b. The code used to generate the SORTP array in the main NPLOT routine, page B12.

c. The SORTLEN subroutine called by the hidden line routine, page B12.

d. The VEC subroutine called by the hidden line routine to plot absolute vectors, pages B13-B14.

```
C *
C *     HIDDEN LINE MODULE:  HIDDEN
C *
C *     This module uses the JONES-D hidden line technique
C *     developed by Gary Jones, NASA/GSFC, Mail Code 731,
C *     Greenbelt, MD, 20771   (Tel: 301 344-7166)
C *     This module is designed to function with the NPLOT
C *     Nastran plot package.
C *
C *
C *     VERSION DATE:  April 20, 1982
C *
C *
C *     PARAMETER DESCRIPTION:
C *
C *     PGRID - Coordinate locations of grid points after rotations
C *             and perspective transformation.
C *
C *     VECLIS - Global vector list, pointers to PGRID.
C *
C *     SURLIS - Global surface list, pointers to PGRID.
C *
C *     NVEC - Number of vectors in the global vector list, VECLIS.
C *
C *     NSUR - Number of surfaces in the global surface list, SURLIS.
C *
C *     X1,Y1,Z1 - View parameters:  2,1,3 = XY view
C *                                  3,2,1 = YZ view
C *                                  1,3,2 = XZ view
C *
C *     VISLIS - Visibility table for grid points. A value of 2 indicates
C *              visibility; passed to the main NPLOT routine to faciltate
C *              subsequent labelling of visible grid points and elements.
C *
C *     XMAX,XMIN,YMAX,YMIN - Horizontal and vertical max and min values.
c *
C *
C *     SORTP - Shell sort parameters, set up in NPLOT main.
C *
C *     SOLIDVV - Counter used to reject highly redundant solid element
C *               edges. Set up is accomplished at time of element load
C *               in NPLOT.
C *
C *     NSVEC - Number of vectors in the global vector list that are
C *             edges of surface or solid elements.
C *
C *     XYS,NUMS,SURF,IBUCKS,XDELTS,YDELTS - Surface data created in this
C *               routine; passed to the main NPLOT routine to facilitate
C *               subsequent labelling of visible elements.
C *
C *     HP - Flag to indicate if the plot file option is selected. If HP=1
C *          a file that captures the on screen plot data is generated.
C *          The HP4 software uses this file to create plots on an HP7580A
C *          pen plotter.
C *
C *     VECTYPE - Counter used to  indicate that vectors result solely
C *               from line element types. If VECTYPE(I)=0 the edge vector
C *               is not a surface or solid element edge.
C *
```

```
      SUBROUTINE HIDDEN(PGRID,VECLIS,SURLIS,NVEC,NSUR,X1,Y1,Z1,VISLIS
     &,YMAX,YMIN,XMAX,XMIN,SORTP,SOLIDVV,NSVEC
     &,XYS,NUMS,SURF,IBUCKS,XDELTS,YDELTS,HP,VECTYPE)
      INTEGER*2 VECLIS(9000,2),SURLIS(9000,4),VISLIS(9000)
     &,XYV(600,13,13),NUMV(13,13),VP(9000,4),XYS(600,13,13),NUMS(13,13)
     &,JUMP,JMAX,J3,J4,SORTP(8),VECTYPE(9000)
     &,X1,Y1,Z1,SOLIDVV(9000),VECVP(9000)
      REAL*4 PGRID(9000,3),VECTOR(15,9000),SURF(26,9000)
     &,XMAX,XMIN,YMAX,YMIN,LINE(1000,4)

      LIMNVEC=600
      LIMNSUR=600
C *
  919 FORMAT(5X,2E15.6)
C *
C *   PREP WORK FOR HIDDEN LINE ****************************************
C *
C *   CALCULATE GRID DENSITIES FOR X-Y BUCKET SORTS
      IF(NSVEC.LT.200) THEN
      IBUCKV=3
      GOTO 3
      ELSE IF (NSVEC.LT.600) THEN
      IBUCKV=6
      GOTO 3
      ELSE IF (NSVEC.LT.1500) THEN
      IBUCKV=8
      GOTO 3
      ELSE IF (NSVEC.LT.2400) THEN
      IBUCKV=10
      GOTO 3
      ELSE
      IBUCKV=13
      ENDIF
    3 XYVFAC=IBUCKV-.01
      IF (NSUR.LT.100) THEN
      IBUCKS=3
      GOTO 4
      ELSE IF (NSUR.LT.600) THEN
      IBUCKS=8
      GOTO 4
      ELSE IF (NSUR.LT.1200) THEN
      IBUCKS=10
      GOTO 4
      ELSE
      IBUCKS=13
      ENDIF
    4 XYSFAC=IBUCKS-.01
      XDIF=(XMAX-XMIN)
      IF(XDIF.EQ.0.0) THEN
      XDIF = .01
      ENDIF
      YDIF = (YMAX-YMIN)
      IF(YDIF.EQ.0.0) THEN
      YDIF= .01
      ENDIF
      XDELTV=XYVFAC/XDIF
      XDELTS=XYSFAC/XDIF
      YDELTV=XYVFAC/YDIF
      YDELTS=XYSFAC/YDIF
C *
C *   ZERO THE CELL LENGTH ARRAYS
```

```
      DO 5 I=1,IBUCKV
      DO 5 J=1,IBUCKV
    5 NUMV(J,I)=0
      DO 6 I=1,IBUCKS
      DO 6 J=1,IBUCKS
    6 NUMS(J,I)=0
C *
C *   BEGIN VECTOR(LINE) PREP ***************
C *
      DO 140 NPT=1,NVEC
      IF(SOLIDVV(NPT).GT.23) GOTO 140
      VECVP(NPT)=1
      N1=VECLIS(NPT,1)
      N2=VECLIS(NPT,2)
      VECTOR(1,NPT)=PGRID(N1,X1)
      VECTOR(2,NPT)=PGRID(N1,Y1)
      VECTOR(3,NPT)=PGRID(N1,Z1)
      VECTOR(4,NPT)=PGRID(N2,X1)
      VECTOR(5,NPT)=PGRID(N2,Y1)
      VECTOR(6,NPT)=PGRID(N2,Z1)
      VECTOR(7,NPT)=VECTOR(5,NPT)-VECTOR(2,NPT)
      VECTOR(8,NPT)=VECTOR(1,NPT)-VECTOR(4,NPT)
      VECTOR(9,NPT)=-VECTOR(7,NPT)*VECTOR(4,NPT)-VECTOR(8,NPT)
     &*VECTOR(5,NPT)
      VECTOR(10,NPT)=MAX(VECTOR(1,NPT),VECTOR(4,NPT))
      VECTOR(11,NPT)=MAX(VECTOR(2,NPT),VECTOR(5,NPT))
      VECTOR(12,NPT)=MAX(VECTOR(3,NPT),VECTOR(6,NPT))
      VECTOR(13,NPT)=MIN(VECTOR(1,NPT),VECTOR(4,NPT))
      VECTOR(14,NPT)=MIN(VECTOR(2,NPT),VECTOR(5,NPT))
      VECTOR(15,NPT)=MIN(VECTOR(3,NPT),VECTOR(6,NPT))
C *
C *   BUCKET SORT (X,Y) ON VECTORS TO PRODUCE VECTOR MAP (XYV)
      ISTX=1+(VECTOR(13,NPT)-XMIN)*XDELTV
      ISPX=1+(VECTOR(10,NPT)-XMIN)*XDELTV
      ISTY=1+(VECTOR(14,NPT)-YMIN)*YDELTV
      ISPY=1+(VECTOR(11,NPT)-YMIN)*YDELTV
C *
      IF(VECTYPE(NPT).GE.1) THEN
      DO 137 J=ISTX,ISPX
      DO 137 I=ISTY,ISPY
      NUMV(J,I)=NUMV(J,I)+1
  137 XYV(NUMV(J,I),J,I)=NPT
      ENDIF
C *
  138 CONTINUE
      VP(NPT,1)=ISTX
      VP(NPT,2)=ISPX
      VP(NPT,3)=ISTY
      VP(NPT,4)=ISPY
  140 CONTINUE
C *
C *   SHELL SORT BY DEPTH (Z) OF VECTOR MAP (XYV)
      DO 145 IX=1,IBUCKV
      DO 145 IY=1,IBUCKV
      JLEN=NUMV(IX,IY)
      IF(JLEN LE 1) GOTO 145
      CALL SORTLEN(JLEN,JCT)
      DO 144 JINDEX=JCT,1,-1
      JUMP=SORTP(JINDEX)
      JMAX=JLEN-JUMP
  146 IFLIP=0
```

```
      DO 143 M=1,JMAX
      N=M+JUMP
      J3=XYV(M,IX,IY)
      J4=XYV(N,IX,IY)
      IF(VECTOR(15,J3).GT.VECTOR(15,J4)) THEN
      XYV(M,IX,IY)=J4
      XYV(N,IX,IY)=J3
      IFLIP=1
      ENDIF
  143 CONTINUE
      IF(IFLIP.EQ.1) GOTO 146
  144 CONTINUE
  145 CONTINUE
C *
C *   CHECK FOR OVERFLOW IN VECTOR MAP, XYV
      DO 149 I=1,IBUCKV
      DO 149 J=1,IBUCKV
      IF(NUMV(J,I).GT.LIMNVEC) THEN
      TYPE 147, LIMNVEC
  147 FORMAT(/,1X,'OVERFLOW IN VECTOR MAP XYV',9X,'LIMIT =',I4)
      GOTO 800
      ENDIF
  149 CONTINUE
C *
C *   BEGIN SURFACE PREP **************
C *
      DO 150 NPT=1,NSUR
      N1=SURLIS(NPT,1)
      N2=SURLIS(NPT,2)
      N3=SURLIS(NPT,3)
      N4=SURLIS(NPT,4)
      XP1=PGRID(N1,X1)
      XP2=PGRID(N2,X1)
      XP3=PGRID(N3,X1)
      XP4=PGRID(N4,X1)
      YP1=PGRID(N1,Y1)
      YP2=PGRID(N2,Y1)
      YP3=PGRID(N3,Y1)
      YP4=PGRID(N4,Y1)
      ZP1=PGRID(N1,Z1)
      ZP2=PGRID(N2,Z1)
      ZP3=PGRID(N3,Z1)
      ZP4=PGRID(N4,Z1)
      SURF(1,NPT)=XP2-XP1
      SURF(2,NPT)=XP3-XP2
      SURF(3,NPT)=XP4-XP3
      SURF(4,NPT)=XP4-XP1
      SURF(5,NPT)=XP1-XP3
      SURF(6,NPT)=YP1-YP2
      SURF(7,NPT)=YP2-YP3
      SURF(8,NPT)=YP3-YP4
      SURF(9,NPT)=YP1-YP4
      SURF(10,NPT)=YP3-YP1
      SURF(11,NPT)=XP2*YP3-XP3*YP2
      SURF(12,NPT)=XP3*YP1-XP1*YP3
      SURF(13,NPT)=XP1*YP2-XP2*YP1
      SURF(14,NPT)=XP3*YP4-XP4*YP3
      SURF(15,NPT)=XP1*YP4-XP4*YP1
      SURF(16,NPT)=MAX(XP1,XP2,XP3,XP4)
      SURF(17,NPT)=MIN(XP1,XP2,XP3,XP4)
```

```
B6      SURF(19,NPT)=MIN(YP1,YP2,YP3,YP4)
        SURF(20,NPT)=MAX(ZP1,ZP2,ZP3,ZP4)
        SURF(21,NPT)=MIN(ZP1,ZP2,ZP3,ZP4)
C *
C *     BUCKET SORT (X,Y) ON SURFACES TO PRODUCE SURFACE MAP (XYS)
        ISTX=1+(SURF(17,NPT)-XMIN)*XDELTS
        ISPX=1+(SURF(16,NPT)-XMIN)*XDELTS
        ISTY=1+(SURF(19,NPT)-YMIN)*YDELTS
        ISPY=1+(SURF(18,NPT)-YMIN)*YDELTS
 335    DO 337 J=ISTX,ISPX
        DO 337 I=ISTY,ISPY
        NUMS(J,I)=NUMS(J,I)+1
 337    XYS(NUMS(J,I),J,I)=NPT
C *
C *     SURFACE AREA CALCULATION
        AREA=ABS(SURF(11,NPT)+SURF(12,NPT)+SURF(13,NPT))
       &+ABS(SURF(12,NPT)+XP1*YP4+XP4*SURF(10,NPT)-XP3*YP4)
        IF(AREA.EQ.0.0) THEN
        SURF(22,NPT)=-1.0
        ELSE
        SURF(22,NPT)=1/AREA
        ENDIF
C *
C *     SET SURFACE ELEMENT TO ZERO
        SURF(25,NPT)=0.0
  150   CONTINUE
C *
C *     SHELL SORT BY DEPTH (Z) OF SURFACE MAP (XYS)
        DO 450 IX=1,IBUCKS
        DO 450 IY=1,IBUCKS
        JLEN=NUMS(IX,IY)
        IF(JLEN.LE.1) GOTO 450
        CALL SORTLEN(JLEN,JCT)
        DO 435 JINDEX=JCT,1,-1
        JUMP=SORTP(JINDEX)
        JMAX=JLEN-JUMP
 446    IFLIP=0
        DO 400 M=1,JMAX
        N=M+JUMP
        J3=XYS(M,IX,IY)
        J4=XYS(N,IX,IY)
        IF(SURF(21,J3).GT.SURF(21,J4)) THEN
        XYS(M,IX,IY)=J4
        XYS(N,IX,IY)=J3
        IFLIP=1
        ENDIF
 400    CONTINUE
        IF(IFLIP.EQ.1) GOTO 446
 435    CONTINUE
 450    CONTINUE
C *
C *     CHECK FOR OVERFLOW IN SURFACE MAP, XYS
        DO 156 I=1,IBUCKS
        DO 156 J=1,IBUCKS
        IF(NUMS(J,I).GT.LIMNSUR) THEN
        TYPE 157,LIMNSUR
  157   FORMAT(/,1X,'OVERFLOW IN SURFACE MAP XYS',9X,'LIMIT =',I4)
        GO TO 800
        ENDIF
  156   CONTINUE
C *
```

```
C *    HIDDEN LINE COMPUTATION ********************************************
C *
       DO 800 I=1,NVEC
       IF(SOLIDVV(I).GT.23) GOTO 800
       FLIP=0
       NSEG=1
       IGPA=VECLIS(I,1)
       IGPB=VECLIS(I,2)
       ICHK=1
       IF(ABS(VECTOR(7,I)).GT.ABS(VECTOR(8,I))) ICHK=2
       IF(VECTOR(7,I).EQ.0.0.AND.VECTOR(8,I).EQ.0.0) GOTO 800
C *    THE ARRAY LINE(I,1-3) HOLDS THE VECTOR END POINTS
C *    AND POINTS OF INTERSECTION. LINE(I,4) INDICATES
C *    SEGMENT VISIBILITY: 0 = HIDDEN, 1 - VISIBLE
       LINE(1,1)=VECTOR(1,I)
       LINE(1,2)=VECTOR(2,I)
       LINE(1,3)=VECTOR(3,I)
       LINE(1,4)=1.
       XMINI=VECTOR(13,I)
       YMINI=VECTOR(14,I)
       XMAXI=VECTOR(10,I)
       YMAXI=VECTOR(11,I)
       ZMAXI=VECTOR(12,I)
C *
C *    LINE INTERSECTION CALCULATION ****************
       DO 610 IX=VP(I,1),VP(I,2)
       DO 610 IY=VP(I,3),VP(I,4)
C *
C *    BINARY SEARCH TO OBTAIN SEARCH DEPTH IN VECTOR MAP
       LEN=NUMV(IX,IY)
       IF(LEN.LT.3) GOTO 615
       MAXX=LEN-1
       LOW=0
       LAST=0
       JPC=LEN/2
       DO 607 WHILE (LAST.NE.JPC)
       LAST=JPC
       J3=XYV(JPC,IX,IY)
       IF(VECTOR(15,J3).LT.ZMAXI) THEN
       J3=XYV(JPC+1,IX,IY)
       IF(VECTOR(15,J3).GE.ZMAXI) GOTO 608
       LOW=JPC
       ELSE
       MAXX=JPC
       ENDIF
       JPC=(LOW+MAXX+1)/2
 607   CONTINUE
C *
C *    DETERMINE LINE INTERSECTIONS
       IF(JPC.EQ.1) GOTO 610
 615   JPC=LEN
 608   DO 609 JP=1,JPC
       J=XYV(JP,IX,IY)
C *    VECVP(J)=0 IF VECTOR J HAS BEEN FOUND TO BE INVISIBLE
       IF(VECVP(J).EQ.0) GOTO 609
       IF(VECTOR(13,J).LE.XMAXI.AND.VECTOR(10,J).GE.XMINI.AND.
      &VECTOR(14,J).LE.YMAXI.AND.VECTOR(11,J).GE.YMINI) THEN
       IF(VECLIS(J,1).EQ.VECLIS(I,1).OR.VECLIS(J,1).EQ.VECLIS(I,2)
      &.OR.VECLIS(J,2).EQ.VECLIS(I,2).OR.VECLIS(J,2).EQ.VECLIS(I,1))
      &GOTO 609
       DENOM=VECTOR(7,I)*VECTOR(8,J)-VECTOR(7,J)*VECTOR(8,I)
```

```
      IF(DENOM.NE.0.0) THEN
      IF(ICHK.EQ.1) THEN
      XINT=(VECTOR(8,I)*VECTOR(9,J)-VECTOR(8,J)*VECTOR(9,I))/DENOM
      IF(XINT.GT.XMAXI.OR.XINT.LT.XMINI) GOTO 609
      VXI=(VECTOR(1,I)-XINT)/VECTOR(8,I)
      YINT=VECTOR(2,I)+VXI*VECTOR(7,I)
      ZINT=VECTOR(3,I)+(VECTOR(6,I)-VECTOR(3,I))*VXI
      ELSE
      YINT=(VECTOR(9,I)*VECTOR(7,J)-VECTOR(7,I)*VECTOR(9,J))/DENOM
      IF(YINT.GT.YMAXI.OR.YINT.LT.YMINI) GOTO 609
      VYI=(VECTOR(2,I)-YINT)/VECTOR(7,I)
      XINT=VECTOR(1,I)+VYI*VECTOR(8,I)
      ZINT=VECTOR(3,I)+(VECTOR(3,I)-VECTOR(6,I))*VYI
      ENDIF
      IF(ABS(VECTOR(8,J)).GT.ABS(VECTOR(7,J))) THEN
      IF(XINT.GT.VECTOR(10,J).OR.XINT.LT.VECTOR(13,J)) THEN
      XITF=.001*(ABS(XINT))
      XIT=XINT-XITF
      IF(XIT.GT.VECTOR(10,J)) GOTO 609
      XIT=XINT+XITF
      IF(XIT.LT.VECTOR(13,J)) GOTO 609
      ENDIF
      ZJNT=VECTOR(3,J)+(VECTOR(3,J)-VECTOR(6,J))*
     &(XINT-VECTOR(1,J))/VECTOR(8,J)
      ELSE IF (VECTOR(7,J).NE.0.0) THEN
      IF(YINT.GT.VECTOR(11,J).OR.YINT.LT.VECTOR(14,J)) THEN
      YITF=.001*(ABS(YINT))
      YIT=YINT-YITF
      IF(YIT.GT.VECTOR(11,J)) GOTO 609
      YIT=YINT+YITF
      IF(YIT.LT.VECTOR(14,J)) GOTO 609
      ENDIF
      ZJNT=VECTOR(3,J)+(VECTOR(3,J)-VECTOR(6,J))*
     &(YINT-VECTOR(2,J))/(-VECTOR(7,J))
      ELSE
      GOTO 609
      ENDIF
C *
C *   DEPTH AT INTERSECTION TEST
      IF(ZJNT.LT.ZINT) THEN
      NSEG=NSEG+1
      LINE(NSEG,1)=XINT
      LINE(NSEG,2)=YINT
      LINE(NSEG,3)=ZINT
      LINE(NSEG,4)=1.
      ENDIF
C *
      ENDIF
      ENDIF
  609 CONTINUE
  610 CONTINUE
C *
C *   INSERT END POINT
      NSEG=NSEG+1
      LINE(NSEG,1)=VECTOR(4,I)
      LINE(NSEG,2)=VECTOR(5,I)
      LINE(NSEG,3)=VECTOR(6,I)
      LINE(NSEG,4)=1.
C *
C *   SORT INTERSECTION LIST FROM MIN TO MAX
      IF(NSEG.LE.3) GOTO 600
```

```
      IF(LINE(1,ICHK).GT.LINE(NSEG,ICHK)) FLIP=1
      IF(NSEG.LE.5) THEN
      JCT=1
      GOTO 621
      ELSE IF (NSEG.LE.13) THEN
      JCT=2
      GOTO 621
      ELSE IF (NSEG.LE.29) THEN
      JCT=3
      GOTO 621
      ELSE IF (NSEG.LE.61) THEN
      JCT=4
      GOTO 621
      ELSE IF(NSEG.LE.125) THEN
      JCT=5
      GOTO 621
      ELSE
      JCT=6
      ENDIF
  621 CONTINUE
      DO 620 JINDEX=JCT,1,-1
      JUMP=SORTP(JINDEX)
      JMAX=NSEG-JUMP
  626 IFLIP=0
      DO 623 M=1,JMAX
      N=M+JUMP
      IF(LINE(M,ICHK).GT.LINE(N,ICHK)) THEN
      DO 628 J3=1,4
      TEMP=LINE(M,J3)
      LINE(M,J3)=LINE(N,J3)
      LINE(N,J3)=TEMP
  628 CONTINUE
      IFLIP=1
      ENDIF
  623 CONTINUE
      IF(IFLIP.EQ.1) GOTO 626
  620 CONTINUE
  600 CONTINUE
C *
C *   COMPUTE VISIBILITY OF LINE SEGMENTS **************
      DO 790 J=2,NSEG
      XMID=(LINE(J-1,1)+LINE(J,1))/2
      YMID=(LINE(J-1,2)+LINE(J,2))/2
      ZMID=(LINE(J-1,3)+LINE(J,3))/2
      IX=1+(XMID-XMIN)*XDELTS
      IY=1+(YMID-YMIN)*YDELTS
C *
C *   BINARY SEARCH TO OBTAIN SEARCH DEPTH IN SURFACE MAP
      LEN=NUMS(IX,IY)
      IF(LEN.LT.3) GOTO 715
      MAXX=LEN-1
      LOW=0
      LAST=0
      JPC=LEN/2
      DO 751 WHILE (LAST.NE.JPC)
      LAST=JPC
      J3=XYS(JPC,IX,IY)
      IF(SURF(21,J3).LT.ZMID) THEN
      J3=XYS(JPC+1,IX,IY)
      IF(SURF(21,J3).GE.ZMID) GOTO 753
      LOW=JPC
```

```
      ELSE
      MAXX=JPC
      ENDIF
      JPC=(LOW+MAXX+1)/2
  751 CONTINUE
      IF(JPC.EQ.1) GOTO 790
  715 JPC=LEN
C *
C *   LINE SEGMENT MID-POINT VISIBILITY TESTS
  753 DO 789 JP=1,JPC
      J3=XYS(JP,IX,IY)
      IF(SURF(16,J3).GE.XMID.AND.SURF(17,J3).LE.XMID.AND.
     &SURF(18,J3).GE.YMID.AND.SURF(19,J3).LE.YMID) THEN
C *
C *   CONTAINMENT TEST
      IF(SURF(22,J3).GT.0.0) THEN
      A1=ABS(XMID*SURF(7,J3)+YMID*SURF(2,J3)+SURF(11,J3))
      A2=ABS(XMID*SURF(8,J3)+YMID*SURF(3,J3)+SURF(14,J3))
      A3=ABS(XMID*SURF(6,J3)+YMID*SURF(1,J3)+SURF(13,J3))
      A4=ABS(XMID*SURF(9,J3)+YMID*SURF(4,J3)+SURF(15,J3))
      ARATIO=(A1+A2+A3+A4)*SURF(22,J3)
      IF(ARATIO.LT.(1.001)) THEN
C *
C *   INITIAL DEPTH TEST
      IF(ZMID.GT.SURF(20,J3)) GOTO 780
C *
C *   IS MID-POINT ON THE SURFACE BOUNDARY?
      IF((A1*SURF(22,J3)).LT.(.0005)) GOTO 789
      IF((A2*SURF(22,J3)).LT.(.0005)) GOTO 789
      IF((A3*SURF(22,J3)).LT.(.0005)) GOTO 789
      IF(SURLIS(J3,1).EQ.SURLIS(J3,4)) GOTO 754
      IF((A4*SURF(22,J3)).LT.(.0005)) GOTO 789
  754 CONTINUE
      IF(IGPA.EQ.SURLIS(J3,1).OR.IGPA.EQ.SURLIS(J3,2).OR.
     &IGPA.EQ.SURLIS(J3,3).OR.IGPA.EQ.SURLIS(J3,4)) THEN
      IF(IGPB.EQ.SURLIS(J3,1).OR.IGPB.EQ.SURLIS(J3,2).OR.
     &IGPB.EQ.SURLIS(J3,3).OR.IGPB.EQ.SURLIS(J3,4)) GOTO 789
      ENDIF
C *
C *   IF SURFACE DATA NOT ALREADY COMPUTED THEN CALCULATE IT
      IF(SURF(25,J3).EQ.0.0) THEN
      SURF(25,J3)=SURF(11,J3)+SURF(12,J3)+SURF(13,J3)
      IF(SURF(25,J3).EQ.0.0) GO TO 789
      ZP1=PGRID(SURLIS(J3,1),Z1)
      ZP2=PGRID(SURLIS(J3,2),Z1)
      ZP3=PGRID(SURLIS(J3,3),Z1)
      SURF(23,J3)=ZP1*SURF(7,J3)+ZP2*SURF(10,J3)+ZP3*SURF(6,J3)
      SURF(24,J3)=ZP1*SURF(2,J3)+ZP2*SURF(5,J3)+ZP3*SURF(1,J3)
      SURF(26,J3)=ZP1*SURF(11,J3)+ZP2*SURF(12,J3)+ZP3*SURF(13,J3)
      ENDIF
C *
C *   DEPTH TEST: IF MID-POINT IS BEHIND SURFACE, THEN VECTOR
C *   SEGMENT IS NOT VISIBLE
      ZSUR=(SURF(26,J3)+XMID*SURF(23,J3)+YMID*SURF(24,J3))/SURF(25,J3)
      IF(ZMID.GT.ZSUR) THEN
  780 LINE(J,4)=0.
      GO TO 790
C *
      ENDIF
      ENDIF
```

```
      ENDIF
  789 CONTINUE
  790 CONTINUE
C *
C *
C *   DRAW VISIBLE LINE SEGMENTS
C *
      VECVP(I)=0
      DO 799 J=2,NSEG
      IF(LINE(J,4).NE.0.) THEN
C *   VEC - A ROUTINE TO DRAW AN ABSOLUTE VECTOR
      CALL VEC(LINE(J-1,1),LINE(J-1,2),LINE(J,1),LINE(J,2))
      VECVP(I)=1
C *   IF HP=1 WRITE A PLOT FILE FOR HP7580A PEN PLOTTER
      IF(HP.EQ.1) THEN
      WRITE(19,919) LINE(J-1,1),LINE(J-1,2)
      WRITE(19,919) LINE(J,1),LINE(J,2)
      ENDIF
      ENDIF
  799 CONTINUE
C *
C *
C *    CHECK AND RECORD VISIBILITY OF END POINTS
C *
      IF(NSEG.LT.3.OR.FLIP.EQ.0) THEN
      IF(LINE(2,4).EQ.1.) THEN
      VISLIS(VECLIS(I,1))=2
      ENDIF
      IF(LINE(NSEG,4).EQ.1.) THEN
      VISLIS(VECLIS(I,2))=2
      ENDIF
      ELSE
      IF(LINE(2,4).EQ.1.) THEN
      VISLIS(VECLIS(I,2))=2
      ENDIF
      IF(LINE(NSEG,4).EQ.1.) THEN
      VISLIS(VECLIS(I,1))=2
      ENDIF
      ENDIF
C *
  800 CONTINUE

      RETURN
      END
```

CALCULATION OF SORTP:

```
C *
C       SORT GRSORT - SHELL METHOD
C *
        SORTP(1)=1
        DO 451 I = 2,8
        SORTP(I)=1+SORTP(I-1)*2
  451 CONTINUE
```

SORTLEN SUBROUTINE:

```
C *
C *
C *  CALCULATES SHELL SORT LENGTH PARAMETER
C *
        SUBROUTINE SORTLEN(NLEN, JCT)
        IF(NLEN.LE.5) THEN
        JCT=1
        GOTO 100
        ENDIF
        IF(NLEN.LE.13) THEN
        JCT=2
        GOTO 100
        ENDIF
        IF(NLEN.LE.29) THEN
        JCT=3
        GOTO 100
        ENDIF
        IF(NLEN.LE.61) THEN
        JCT=4
        GOTO 100
        ENDIF
        IF(NLEN.LE.125) THEN
        JCT=5
        GOTO 100
        ENDIF
        IF(NLEN.LE.253) THEN
        JCT=6
        GOTO 100
        ENDIF
        IF(NLEN.LE.510) THEN
        JCT=7
        GOTO 100
        ENDIF
        JCT=8
  100   CONTINUE
        RETURN
        END
```

## VEC SUBROUTINE:



```
C *
C *
C *     VEC: VECTOR DRAW ROUTINE                    Gary Jones/GSFC
C *                                                 December 16, 1981
C *
C *     PLOT10 requires two calls per vector, 'CALL MOVEA(GX1,GY1)'
C *     and 'CALL DRAWA (GX2,GY2)'. This routine combines these
C *     operations into just one call with about a 20 percent reduction
C *     in run time.  The calls in this subroutine are to PLOT10.
C *
      SUBROUTINE VEC(GX1,GY1,GX2,GY2)
      DIMENSION BUFIN(4),BFOUT(4)
      COMMON /TKTRNX/ TMINVX,TMINVY,TMAXVX,TMAXVY,TREALX,TREALY,
     + TIMAGX,TIMAGY,TRCOSF,TRSINF,TRSCAL,TRFACX,TRFACY,
     + TRPAR1,TRPAR2,TRPAR3,TRPAR4,TRPAR5,TRPAR6,KMOFLG(2),
     + KGNMOD,KPADV,KACHAR,KOBLEN,KTRAIL,KLEVEL,KPAD2,
     + KBAUDR,KGNFLG,KGRAFL,KHOMEY,KKMODE,KHORSZ,KVERSZ,KTBLSZ,
     + KSIZEF,KLMRGN,KRMRGN,KFACTR,KTERM,KLINE,KZAXIS,KBEAMX,KBEAMY,
     + KMOVEF,KPCHAR(5),KDASHT,KMINSX,KMINSY,KMAXSX,KMAXSY,KEYCON,
     + KINLFT,KOTLFT,KUNIT
      EQUIVALENCE (BFOUT(1),CXS),(BFOUT(2),CYS),(BFOUT(3),CXE),
     +(BFOUT(4),CYE)
C *
C * THIS SECTION REQUIRED FOR GENISCO G-1000 TERMINAL
C *
      CALL BUFFPK(0,ITEMP)
C *
C * SET UP OPERATIONS
C *
      IF (KGRAFL.EQ.0) THEN
      TREALX=TMINVX+(FLOAT(KBEAMX-KMINSX)/TRFACX)
      TREALY=TMINVY+(FLOAT(KBEAMY-KMINSY)/TRFACY)
      TIMAGX=TREALX
      TIMAGY=TREALY
      KGRAFL=1
      ENDIF
      CALL BUFFPK(1,29)
      KMOVEF=1
C *
C * PROCESS START POINT (GX1,GY1)
C *
      IF(GX1.GT.TMAXVX.OR.GX1.LT.TMINVX.OR.
     +GY1.GT.TMAXVY.OR.GY1.LT.TMINVY) THEN
      CALL XYCNVT(KBEAMX,KBEAMY)
      ELSE
      DX=GX1-TMINVX
      DY=GY1-TMINVY
      IX=IFIX(DX*TRFACX+.5)+KMINSX
      IY=IFIX(DY*TRFACY+.5)+KMINSY
      TREALX=GX1
      TREALY=GY1
      CALL XYCNVT(IX,IY)
      ENDIF
C *
C * PROCESS END POINT (GX2,GY2)
C *
      BUFIN(1)=GX1
      BUFIN(2)=GY1
      BUFIN(3)=GX2
      BUFIN(4)=GY2
```

```
      CALL CLIPT(BUFIN,BFOUT)
      IF (%GNFLG.NE.1) THEN
      IF (CXS.NE.TREALX.OR.CYS.NE.TREALY) THEN
      MODE=KKMODE
      CALL BUFFPK(1,29)
      KMOVEF=1
      DX=CXS-TMINVX
      DY=CYS-TMINVY
      IX=IFIX(DX*TRFACX+.5)+KMINSX
      IY=IFIX(DY*TRFACY+.5)+KMINSY
      CALL XYCNVT(IX,IY)
      KKMODE=MODE
      ENDIF
      DX=CXE-TMINVX
      DY=CYE-TMINVY
      IX=IFIX(DX*TRFACX+.5)+KMINSX
      IY=IFIX(DY*TRFACY+.5)+KMINSY
      CALL XYCNVT(IX,IY)
      TREALX=CXE
      TREALY=CYE
      ENDIF
C *
      RETURN
      END
```

END

DATE

FILMED

JAN 28 1983